09-05-07

Attorney's Docket No. 042933/298965

PATENT

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re:	Nativadade Lobo	Confirmation No.: 5615
Appl. No.:	09/625,201	
Filed:	July 21, 2000	
For:	PULSE SHAPING WHICH COMPENSATES FOR COMPONENT DISTORTION	

BOX ISSUE FEE
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

SUBMITTAL OF PRIORITY DOCUMENT

To complete the requirements of 35 U.S.C. § 119, enclosed is a certified copy of Great Britain priority Application No. 9805126.1, filed March 11, 1998.

Respectfully submitted,

Cory C. Davis

Cory C. Davis
Registration No. 59,932

Customer No. 00826
Alston & Bird LLP
Bank of America Plaza
101 South Tryon Street, Suite 4000
Charlotte, NC 28280-4000
Tel Charlotte Office (704) 444-1000
Fax Charlotte Office (704) 444-1111

"Express Mail" mailing label number EV521113148US
Date of Deposit September 4, 2007

I hereby certify that this paper or fee is being deposited with the United States Postal Service "Express Mail Post Office to Addressee" service under 37 CFR 1.10 on the date indicated above and is addressed to:
BOX ISSUE FEE, Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450

Tamara Stevens

Tamara Stevens

tellectual
roperty
Office

ativity and Innovation

Concept House
Cardiff Road
Newport
South Wales
NP10 8QQ

the undersigned, being an officer duly authorised in accordance with Section 74(1) and (4) the Deregulation & Contracting Out Act 1994, to sign and issue certificates on behalf of Comptroller-General, hereby certify that annexed hereto is a true copy of the documents originally filed in connection with patent application GB9805126.1 filed on 11 March 98.

so certify that the attached copy of the request for grant of a Patent (Form 1/77) bears an ndment, effected by this office, following a request by the applicant and agreed to by the nptroller-General.

cordance with the Patents (Companies Re-registration) Rules 1982, if a company named is certificate and any accompanying documents has re-registered under the Companies 980 with the same name as that with which it was registered immediately before re- ration save for the substitution as, or inclusion as, the last part of the name of the words ic limited company" or their equivalents in Welsh, references to the name of the any in this certificate and any accompanying documents shall be treated as references to me with which it is so re-registered.

rdance with the rules, the words "public limited company" may be replaced by p.l.c., .C. or PLC.

nder the Companies Act does not constitute a new legal entity but merely any to certain additional company law rules.

Signed

Dated 22 August 2007

THE PATENT OFFICE
A
11 MAR 1998
RECEIVED BY FAX

Patents Form 1/77

Patents Act 1977
(Rule 16)

The Patent Office

9
11MAR98 E344686-1 D02716
P01/7700 25.00 - 9805126.1

Request for grant of a patent

(See the notes on the back of this form. You can also get an explanatory leaflet from the Patent Office to help you fill in this form)

The Patent Office

Cardiff Road
Newport
Gwent NP9 1RH

1. Your reference

PAT 98006 GB

2. Patent application number
(The Patent Office will fill in this part)

11 MAR 1998

9805126.1

3. Full name, address and postcode of the or of each applicant *(underline all surnames)*

NOKIA MOBILE PHONES LIMITED
KEILALAHDENTIE 4
02150 ESPOO
FINLAND

Patents ADP number *(if you know it)*

5911995004

If the applicant is a corporate body, give the country/state of its incorporation

4. Title of the invention

A TRANSCEIVER

(confirmed by telephone - letter to follow)

5. Name of your agent *(if you have one)*

"Address for service" in the United Kingdom to which all correspondence should be sent *(including the postcode)*

MRS HELEN LOUISE HAWS
NOKIA MOBILE PHONES
PATENT DEPARTMENT
ST GEORGES COURT
ST GEORGES ROAD
CAMBERLEY
SURREY GU15 3QZ UK

Patents ADP number *(if you know it)*

6944553900 1

6. If you are declaring priority from one or more earlier patent applications, give the country and the date of filing of the or of each of these earlier applications and *(if you know it)* the or each application number

Country	Priority application number *(if you know it)*	Date of filing *(day / month / year)*

7. If this application is divided or otherwise derived from an earlier UK application, give the number and the filing date of the earlier application

Number of earlier application	Date of filing *(day / month / year)*

8. Is a statement of inventorship and of right to grant of a patent required in support of this request? *(Answer 'Yes' if:*

a) any applicant named in part 3 is not an inventor, or
b) there is an inventor who is not named as an applicant, or
c) any named applicant is a corporate body.
See note (d))

YES

Patents Form 1/77

9. Enter the number of sheets for any of the following items you are filing with this form. Do not count copies of the same document

Continuation sheets of this form

Description 5

Claim(s) 1

Abstract

Drawing(s) included in description

10. If you are also filing any of the following, state how many against each item.

Priority documents

Translations of priority documents

Statement of inventorship and right to grant of a patent *(Patents Form 7/77)*

Request for preliminary examination and search *(Patents Form 9/77)*

Request for substantive examination *(Patents Form 10/77)*

Any other documents *(please specify)*

11. I/We request the grant of a patent on the basis of this application.

Signature H Haws Date 11.3.1998

H L HAWS - Agent for the Applicant

12. Name and daytime telephone number of person to contact in the United Kingdom

Mrs H Haws
01276 419346

Warning

After an application for a patent has been filed, the Comptroller of the Patent Office will consider whether publication or communication of the invention should be prohibited or restricted under Section 22 of the Patents Act 1977. You will be informed if it is necessary to prohibit or restrict your invention in this way. Furthermore, if you live in the United Kingdom, Section 23 of the Patents Act 1977 stops you from applying for a patent abroad without first getting written permission from the Patent Office unless an application has been filed at least 6 weeks beforehand in the United Kingdom for a patent for the same invention and either no direction prohibiting publication or communication has been given, or any such direction has been revoked.

Notes

a) If you need help to fill in this form or you have any questions, please contact the Patent Office on 0645 500505.

b) Write your answers in capital letters using black ink or you may type them.

c) If there is not enough space for all the relevant details on any part of this form, please continue on a separate sheet of paper and write "see continuation sheet" in the relevant part(s). Any continuation sheet should be attached to this form.

d) If you have answered 'Yes' Patents Form 7/77 will need to be filed.

e) Once you have filled in the form you must remember to sign and date it.

f) For details of the fee and ways to pay please contact the Patent Office.

■ CDMA Patent

The process at the transmitter of CDMA is roughly

BitStream –> Frame Builder –>**GoldCode Encoding** –> Modulator [**Using LookUp Table**]
D/A –> Filter[Switched Capacitor,**Clock Frequency**] –> Filter[**Resistors,Capasitors**] –> Transmitter

The corresponding process for CDMA receiver is

Complex Number Seq –> **CDMA Process** –> Channel Estimation –> Demodulation [Using Viterbi] –> BitStream

These process are in fact similar to GSM so we could build a dual mode phone. We describe the GSM processes

The transmitter processes are listed below

BitStream –> Frame Builder –> Modulator [**Using LookUp Table**]
D/A –> Filter[Switched Capacitor,**Clock Frequency**] –> Filter[**Resistors,Capasitors**] –> Transmitter

At the receiver we have the reverse process

Complex Number Seq –> **Multiplying incomming stream by i^n –> BitSinchronisation** –> Channel Estimation –> Demodulation [**Using Viterbi**] –> BitStream

Note
1)The mode dependent process are highlighted.
2) The same filter can be used in both the cases if the clock frequencies are selected with care.
3)The timing issues, and frequency dependent items like antennas have been ignored in this paper.

4)In this paper we use many different pulses depending on the bandwidth, chip rate, required and we trade off the non constant amplitude. This is the key inovation (to get higher bit rates).
The selection of bitrate is just a matter of engineering design involving BER, Power Amplifier efficiency and spectrum utilisation. We also point out that although in this paper we have designed a modulator from first principles, we would in pratice use the current Nokia Mobile Phones lookup table architecure [based on a pulse width of 4T] or an architecure based on 6T pulse width obtaining the correct syclostationary spectrum in each mode by the appropiate filter [Analog]. The Switched Capacitor will in pratice do most of the spectral shaping common to both modes. This again is standard to design. Again the key inovation is the fact that we use one and/or two or more not n–Laurent pulses $C_0(t)$, $C_1(t)$ . . . in the construction derived through an optimisation process to construct the modulator. Others have used, $C_0(t)$ only, varing at most values of $B_b T$ to vary the pulse (Linearised GMSK) , or prefiltering to compensate for the reconstruction filter, we have proposed to use two pulses (designed using an optimiser), to get lower amplitude variation.

5)The key inovative step is the nature of the functions used to transform the gold code into the sequence used in the correlator by the function below. It is these functions that we wish to protect in the patent. The transformation need only occur once,and the results stored, as the mobile can pre calculate the sequence d_i for i = 0,1, .. N–1 given the code $\{c_0, c_1, \ldots . c_{N-1}\}$ when the code is asigned.

Transformation 1 (to detect +1 symbol, C_0 equivalent pulse)

$y_i = (-1)^i$ for $i = 0, 1, 2, \ldots N-1$
Given code $\{c_0, c_1, \ldots c_{N-1}\}$ where N is the number of elements in the sequence.
$a_i = 1$ if $c_i = 1$; for $i = 0, \ldots N-1$
$a_i = -1$ if $c_i = 0$; for $i = 0, \ldots N-1$
$b_0 = a_0$;
$b_i = b_{i-1} + a_i$ for $i = 1, 2, \ldots N-1$;
$d_i = y_i\, i^{b_i}$ for $i = 0, 1, 2, \ldots N-1$ and
$i = \sqrt{(-1)}$
There is another transformation that can also be used
Transformation 1 b (to detect + 1 symbol, C_0 equivalent pulse) using the same notation for d_i
$d_i = i^{-b_i}$ for $i = 0, 1, 2, \ldots N-1$ and

Transformation 2 (to detect −1 symbol, C_0 equivalent pulse)
$y_i = (-1)^i$ for $i = 0, 1, 2, \ldots N-1$
Given code $\{c_0, c_1, \ldots c_{N-1}\}$ where N is the number of elements in the sequence.
$a_i = -1$ if $c_i = 1$; for $i = 0, \ldots N-1$
$a_i = 1$ if $c_i = 0$; for $i = 0, \ldots N-1$
$b_0 = a_0$;
$b_i = b_{i-1} + a_i$ for $i = 1, 2, \ldots N-1$;
$d_i = y_i\, i^{b_i}$ for $i = 0, 1, 2, \ldots N-1$ and
$i = \sqrt{(-1)}$
There is another transformation that can also be used
Transformation 2 b (to detect − 1 symbol, C_0 equivalent pulse)
$d_i = i^{-b_i}$ for $i = 0, 1, 2, \ldots N-1$

Transformation 3 (to detect +1 symbol C_1 equivalent pulse)
$y_i = (-1)^i$ for $i = 0, 1, 2, \ldots N-1$
Given code $\{c_0, c_1, \ldots c_{N-1}\}$ where N is the number of elements in the sequence.
$a_i = 1$ if $c_i = 1$; for $i = 0, \ldots N-1$
$a_i = -1$ if $c_i = 0$; for $i = 0, \ldots N-1$
$b_0 = a_0 - /+ a_N$;
$b_i = b_{i-1} + a_i - a_{i-1}$ for $i = 1, 2, \ldots N-1$;
$d_i = y_i\, i^{b_i}$ for $i = 0, 1, 2, \ldots N-1$ and
$i = \sqrt{(-1)}$
There is another transformation that can also be used
Transformation 3 b (to detect 1 symbol, C_1 equivalent pulse)
$d_i = i^{-b_i}$ for $i = 0, 1, 2, \ldots N-1$ and

Transformation 4 (to detect −1 symbol C_1 equivalent pulse)
$y_i = (-1)^i$ for $i = 0, 1, 2, \ldots N-1$
Given code $\{c_0, c_1, \ldots c_{N-1}\}$ where N is the number of elements in the sequence.
$a_i = -1$ if $c_i = 1$; for $i = 0, \ldots N-1$
$a_i = 1$ if $c_i = 0$; for $i = 0, \ldots N-1$
$b_0 = a_0 - /+ a_N$;
$b_i = b_{i-1} + a_i - a_{i-1}$ for $i = 1, 2, \ldots N-1$;
$d_i = y_i\, i^{b_i}$ for $i = 0, 1, 2, \ldots N-1$ and
$i = \sqrt{(-1)}$
There is another transformation that can also be used
Transformation 4 b (to detect − 1 symbol, C_1 equivalent pulse)
$d_i = i^{-b_i}$ for $i = 0, 1, 2, \ldots N-1$ and

We also claim any
a)cyclic rotation of the sequences resulting from Transformations 1,2,3,4 and 1b,2b,3b,4b
b)multiplication of the elements of the sequence obtained from Transformations 1,2,3,4 and 1b,2b,3b,4b by a constant (real or complex)

are also claimed to be transformations that we wish to patent.
We also claim any combination of cyclic rotation followed by linear scaling of the result is patented.

The subroutine CodingTransformation implements the transformations 1,2,3,4

We also stress that while the bitstream is differentially encoded and the substitution used is $\{0 \to 1, 1 \to -1\}$ the substitution elsewhere used for the gold codes is $\{0 \to -1, 1 \to 1\}$

6)The demodulation in the CDMA mode uses the 4 separately transformed codes(Transformation 1,2,3,4) or (Transformation 1,2). Also we point out that in this scheme we have two adjacent chip positions that generate larg values of correlation. With thus using 2 pulses and 8 complex correlations, we can obtain, should we desire the best possible demodulation. Thus, most of the work is done by the gold code. during correlation.
7)There is nothing special about the Gold Code used in the example. Any gold like code will do as long as it posseses the usual properties of a gold code.

8) In case we consider a BT = 0.15 etc GSM modulation scheme, (i.e. twice the speed) we can keep the

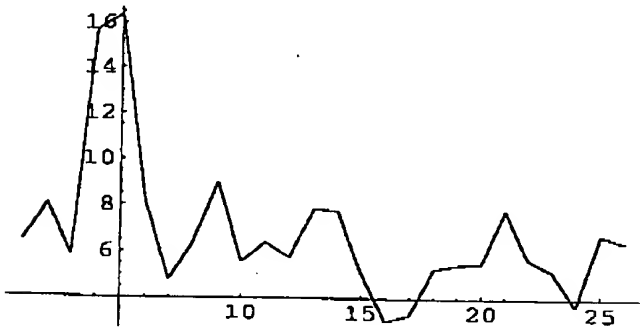
Example of the output using different encoders which encode transformations to correlate with the input of a bipolar bit stream of the first training sequence of GSM system where we have used the rule {0 –> –1} and not differentially encoded the .

The first training sequence of GSM system where we have used the rule $\{0 \to -1\}$ and not differentially encoded it is calles GSMseq ={−1,−1,1,−1,−1,1,−1,1,1,1,1,−1,−1,−1,−1,1,−1,−1,−1,1,1,−1,−1,1,−1,1,1,1,1}

```
ModOutputGSM = Modulator[L][GSMseq, NumberOfCurves → 2,
  ModulatingPulse → OptPulseScaled, SamplingInterval → T / 4];
```

We call the Modulator output ModOutputGSM. PrimitiveCDMAReceiver encodes GSMseq according to transformation rule 1

```
TomGSM1 = PrimitiveCDMAReceiver[ModOutputGSM, GSMseq, 1, 4];

ListPlot[TomGSM1 // Abs, PlotJoined -> True, PlotRange -> All]
```

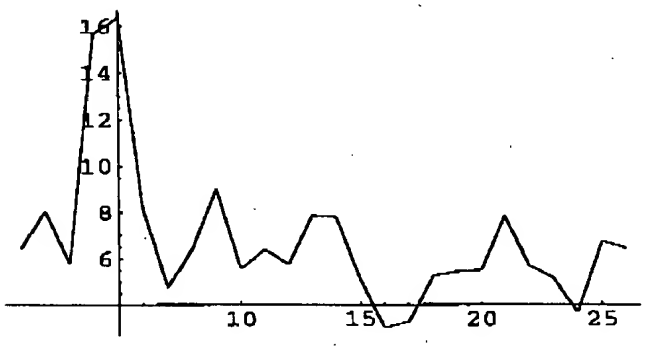


- Graphics -

We have plotted the autocorrelation. The next graph shows the autocorrelation when GSMseq is encoded with transformation 1b. It seems the same as with transformation 1b

```
PrimitiveCDMAReceiver2[ModOutputGSM, GSMseq, 1, 4];
```

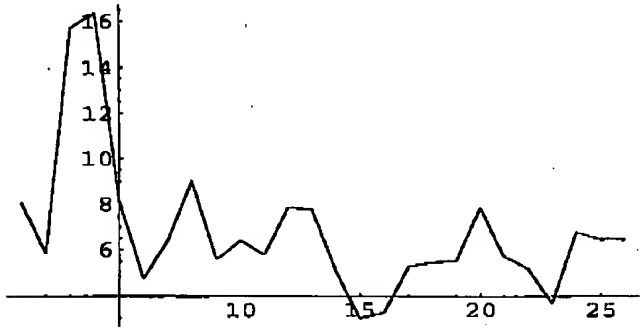
```
ListPlot[% // Abs, PlotJoined -> True, PlotRange -> All]
```




```
- Graphics -
```

PrimitiveCDMAReceiverGSM2Pulse encodes GSMseq with Transform 3

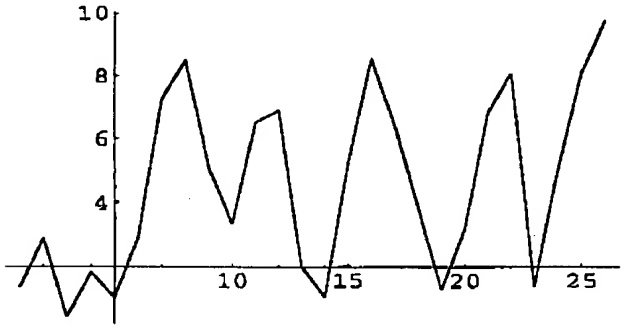
```
PrimitiveCDMAReceiverGSM2Pulse[ModOutputGSM, GSMseq, 1, 4];
ListPlot[% // Abs, PlotJoined -> True, PlotRange -> All]
```




```
- Graphics -
```

PrimitiveCDMAReceiverGSM2PulseEfficient encodes GSMseq with Transform3b. This seems to be diffrent

```
PrimitiveCDMAReceiverGSM2PulseEfficient[ModOutputGSM, GSMseq, 1, 4];
ListPlot[% // Abs, PlotJoined -> True, PlotRange -> All]
```




```
- Graphics -
```

■ Dual Mode CDMA and GSM Operation

In this work we demonstrate how to build a dual mode phone using as much as possible the same hardware for both the systems.
Given the following process for GSM Transmitter

BitStream –> Frame Builder –> Modulator **[Using LookUp Table]**
D/A –> Filter[Switched Capacitor,**Clock Frequency**] –> Filter[**Resistors,Capasitors**] –> Transmitter

At the receiver we have the reverse process

Complex Number Seq –> Channel Estimation –> Demodulation **[Using Viterbi]** –> BitStream

The corresponding process at the CDMA transmitter is roughly

BitStream –> Frame Builder –>**GoldCode Encoding** –> Modulator **[Using LookUp Table]**
D/A –> Filter[Switched Capacitor,**Clock Frequency**] –> Filter[**Resistors,Capasitors**] –> Transmitter

The corresponding process for CDMA receiver is

Complex Number Seq –> **CDMA Process** –> Channel Estimation –> Demodulation **[Using Binary Decision]** –> BitStream

Note
1)The mode dependent process are highlighted.
2) The same filter can be used in both the cases if the clock frequencies are selected with care.
3)The timing issues, and frequency dependent items like antennas have been ignored in this paper.

4)In this paper we use many different pulses depending on the bandwidth, chip rate, required and we trade off the non constant amplitude. This is the key inovation (to get higher bit rates).
The selection of bitrate is just a matter of engineering design involving BER, Power Amplifier efficiency and spectrum utilisation. We also point out that although in this paper we have designed a modulator from first principles, we would in pratice use the current Nokia Mobile Phones lookup table architecure [based on a pulse width of 4T] or an architecure based on 6T pulse width obtaining the correct syclostatioary spectrum in each mode by the appropiate filter [Analog]. The Switched Capacitor will in pratice do most of the spectral shaping common to both modes. This again is standard to design. Again the key inovation is the fact that we use one and/or two or more not n–Laurent pulses $C_0(t)$, $C_1(t)$ . . . in the construction derived through an optimisation process to construct the modulator. Others have used, $C_0(t)$ only, varing at most values of $B_b T$ to vary the pulse (Linearised GMSK) , or prefiltering to compensate for the reconstruction filter, we have proposed to use two pulses (designed using an optimiser), to get lower amplitude variation.

5)The key inovative step is the nature of the functions used to transform the gold code into the sequence used in the correlator by the function below. It is these functions that we wish to protect in the patent. The transformation need only occur once,and the results stored, as the mobile can pre calculate the sequence d_i for i = 0,1, .. N–1 given the code $\{c_0, c_1, \ldots c_{N-1}\}$ when the code is asigned.

Transformation 1 (to detect +1 symbol, C_0 equivalent pulse)

$y_i = (-1)^i$ for $i = 0, 1, 2, \ldots N-1$
Given code $\{c_0, c_1, \ldots c_{N-1}\}$ where N is the number of elements in the sequence.
$a_i = 1$ if $c_i = 1$; for $i = 0, \ldots N-1$
$a_i = -1$ if $c_i = 0$; for $i = 0, \ldots N-1$
$b_0 = a_0$;

$b_i = b_{i-1} + a_i$ for $i = 1, 2, \ldots N-1$;
$d_i = y_i\, i^{b_i}$ for $i = 0, 1, 2, \ldots N-1$ and
$i = \sqrt{(-1)}$

There is another transformation that can also be used although it does not perform as well

Transformation 1 b (to detect + 1 symbol, C_0 equivalent pulse)

$d_i = i^{-b_i}$ for $i = 0, 1, 2, \ldots N-1$ and

Transformation 2 (to detect −1 symbol,C_0 equivalent pulse)

$y_i = (-1)^i$ for $i = 0, 1, 2, \ldots N-1$
Given code { $c_0, c_1, \ldots c_{N-1}$ } where N is the number of elements in the sequence .
$a_i = -1$ if $c_i = 1$; for $i = 0, \ldots N-1$
$a_i = 1$ if $c_i = 0$; for $i = 0, \ldots N-1$
$b_0 = a_0$;
$b_i = b_{i-1} + a_i$ for $i = 1, 2, \ldots N-1$;
$d_i = y_i\, i^{b_i}$ for $i = 0, 1, 2, \ldots N-1$ and
$i = \sqrt{(-1)}$

There is another transformation that can also be used although it does not perform as well

Transformation 2 b (to detect − 1 symbol, C_0 equivalent pulse)

$d_i = i^{-b_i}$ for $i = 0, 1, 2, \ldots N-1$ and

Transformation 3 (to detect +1 symbolC_1 equivalent pulse)

$y_i = (-1)^i$ for $i = 0, 1, 2, \ldots N-1$
Given code { $c_0, c_1, \ldots c_{N-1}$ } where N is the number of elements in the sequence .
$a_i = 1$ if $c_i = 1$; for $i = 0, \ldots N-1$
$a_i = -1$ if $c_i = 0$; for $i = 0, \ldots N-1$
$b_0 = a_0 - /+ a_N$;
$b_i = b_{i-1} + a_i - a_{i-1}$ for $i = 1, 2, \ldots N-1$;
$d_i = y_i\, i^{b_i}$ for $i = 0, 1, 2, \ldots N-1$ and
$i = \sqrt{(-1)}$

There is another transformation that can also be used although it does not perform as well

Transformation 3 b (to detect 1 symbol, C_1 equivalent pulse)

$d_i = i^{-b_i}$ for $i = 0, 1, 2, \ldots N-1$ and

Transformation 4 (to detect −1 symbolC_1 equivalent pulse)

$y_i = (-1)^i$ for $i = 0, 1, 2, \ldots N-1$
Given code { $c_0, c_1, \ldots c_{N-1}$ } where N is the number of elements in the sequence .
$a_i = -1$ if $c_i = 1$; for $i = 0, \ldots N-1$
$a_i = 1$ if $c_i = 0$; for $i = 0, \ldots N-1$
$b_0 = a_0 - /+ a_N$;
$b_i = b_{i-1} + a_i - a_{i-1}$ for $i = 1, 2, \ldots N-1$;
$d_i = y_i\, i^{b_i}$ for $i = 0, 1, 2, \ldots N-1$ and
$i = \sqrt{(-1)}$

There is another transformation that can also be used although it does not perform as well

Transformation 4 b (to detect − 1 symbol, C_i equivalent pulse)

$d_i = i^{-b_i}$ for $i = 0, 1, 2, \ldots N-1$ and

We also claim any
a)cyclic rotation of the sequences resulting from Transformations 1,2,3,4 and 1b,2b,3b,4b
b)multiplication of the elements of the sequence obtained from Transformations 1,2,3,4 and 1b,2b,3b,4b by a constant (real or complex)
are also claimed to be transformations that we wish to patent.
We also claim any combination of cyclic rotation followed by linear scaling of the result is patented.

The subroutine CodingTransformation implements the transformations 1,2,3,4

We also stress that while the bitstream is differentially encoded and the substitution used is $\{0 \to 1, 1 \to -1\}$ the substitution elsewhere used for the gold codes is $\{0 \to -1, 1 \to 1\}$

6)The demodulation in the CDMA mode uses the 4 separately transformed codes(Transformation 1,2,3,4) or (Transformation 1,2). With thus using 2 pulses and 4 at most complex correlations, we can demodulate the bitstream. Thus, most of the work is done by the gold code. during correlation.
7)There is nothing special about the Gold Code used in the example. Any gold like code will do as long as it posseses the usual properties of a gold code.

8) In case we consider a BT = 0.15 etc GSM modulation scheme, (i.e. twice the speed) we can keep the

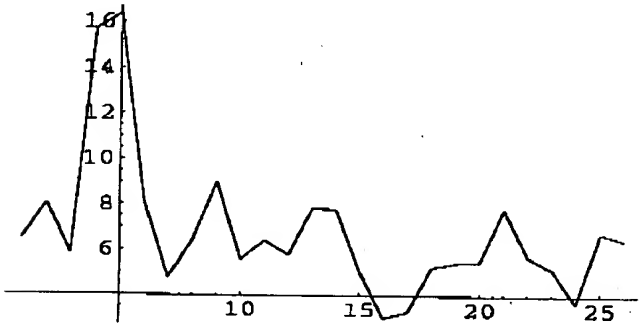
Example of the output using PrimitiveCDMAReceiverGSM2Pulse which encodes transformation 2 to correlate with the input of a bipolar bit stream of the first training sequence of GSM system where we have used the rule $\{0 \to -1\}$ and not differentially encoded the .

The first training sequence of GSM system where we have used the rule $\{0 \to -1\}$ and not differentially encoded it is calles GSMseq = $\{-1,-1,1,-1,-1,1,1,-1,1,1,1,1,-1,-1,1,-1,1,1,-1,1,-1,1,1,-1,1,-1,1,1,1,1\}$

```
ModOutputGSM = Modulator[L][GSMseq, NumberOfCurves → 2,
  ModulatingPulse → OptPulseScaled, SamplingInterval → T / 4];
```

We call the Modulator output ModOutputGSM. PrimitiveCDMAReceiver encodes GSMseq according to transformation rule 1

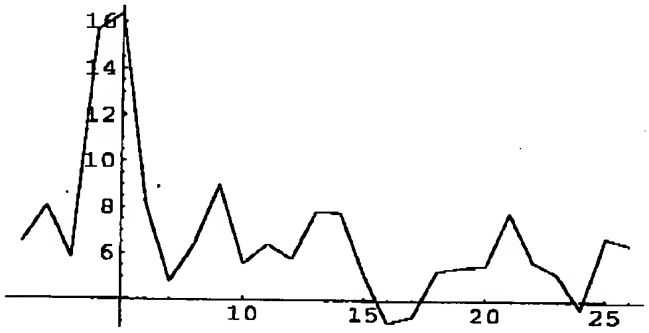
```
TomGSM1 = PrimitiveCDMAReceiver[ModOutputGSM, GSMseq, 1, 4];

ListPlot[TomGSM1 // Abs, PlotJoined -> True, PlotRange -> All]
```



- Graphics -

We have plotted the autocorrelation. The next graph shows the autocorrelation when GSMseq is encoded with transformation 1b. It seems the same as with transformation 1b

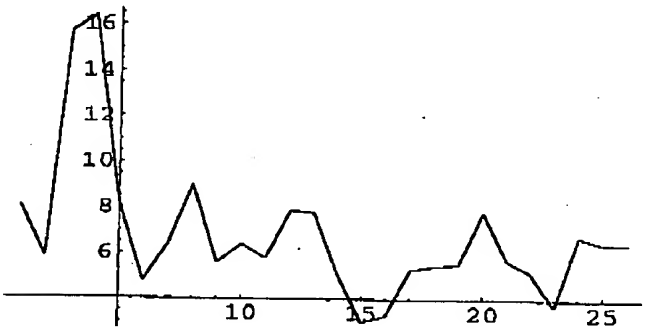
```
PrimitiveCDMAReceiver2[ModOutputGSM, GSMseq, 1, 4];

ListPlot[% // Abs, PlotJoined -> True, PlotRange -> All]
```




```
- Graphics -
```

PrimitiveCDMAReceiverGSM2Pulse encodes GSMseq with transform 3

```
PrimitiveCDMAReceiverGSM2Pulse[ModOutputGSM, GSMseq, 1, 4];

ListPlot[% // Abs, PlotJoined -> True, PlotRange -> All]
```

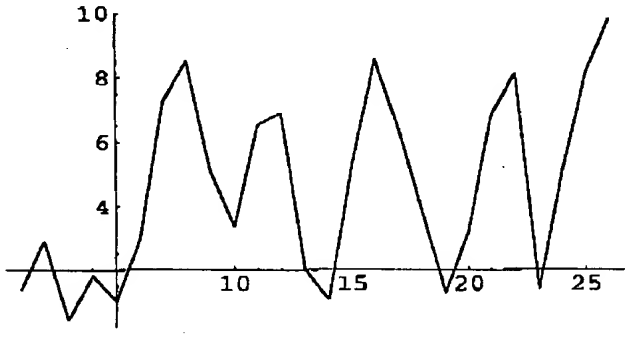



```
- Graphics -
```

PrimitiveCDMAReceiverGSM2PulseEfficient encodes GSMseq with transform3b

```
PrimitiveCDMAReceiverGSM2PulseEfficient[ModOutputGSM, GSMseq, 1, 4];
```

```
ListPlot[% // Abs, PlotJoined -> True, PlotRange -> All]
```




```
- Graphics -
```

■ GSM with BT = 0.15

We can perform the current modulation scheme using a BT = 0.15. The problem occurs with the receiver. The linearised approx does not work well with low BT products. We have to some how incorporate the energy of the second Laurent pulse $C_{1,}$ into the Viterbi.

We propose useing the same frame structure without differentially encoding the bits. We make the transformation {0–> −1}.

We describe the current GSM processes

1)The key inovation is **not differentially** encoding the data but keeping the same frame structure. We use the current or new longer m–sequences.

2)The other key inovative step is the nature of the functions used to transform the training sequences into the sequence used in the correlation to get the impulse respose in the Viterbi. They are listed below. It is these functions that we wish to protect in the patent. The transformation need only occur once,and the results stored, as the mobile can pre calculate the sequence d_i for i = 0,1, .. N−1 given the training sequence $\{c_0, c_1, \ldots c_{N-1}\}$. Here N = 26

Transformation 1 (to detect +1 symbol, C_0 equivalent pulse)

$y_i = (-1)^i$ for $i = 0, 1, 2, \ldots . N-1$

Given code $\{c_0, c_1, \ldots c_{N-1}\}$ where N is the number of elements in the sequence .

$a_i = 1$ if $c_i = 1$; for $i = 0, \ldots N-1$

$a_i = -1$ if $c_i = 0$; for $i = 0, \ldots N-1$

$b_0 = a_0$;

$b_i = b_{i-1} + a_i$ for $i = 1, 2, \ldots . N-1$;

$d_i = y_i\, i^{b_i}$ for $i = 0, 1, 2, \ldots . N-1$ and

$i = \sqrt{(-1)}$

There is another transformation that can also be used

Transformation 1 b (to detect + 1 symbol, C_0 equivalent pulse) using the same notation for d_i

$d_i = i^{-b_i}$ for $i = 0, 1, 2, \ldots . N-1$ and

Transformation 2 (to detect −1 symbol,C_0 equivalent pulse)

$y_i = (-1)^i$ for $i = 0, 1, 2, \ldots . N-1$

Given code $\{c_0, c_1, \ldots c_{N-1}\}$ where N is the number of elements in the sequence .

$a_i = -1$ if $c_i = 1$; for $i = 0, \ldots N-1$

$a_i = 1$ if $c_i = 0$; for $i = 0, \ldots N-1$

$b_0 = a_0$;

$b_i = b_{i-1} + a_i$ for $i = 1, 2, \ldots . N-1$;

$d_i = y_i\, i^{b_i}$ for $i = 0, 1, 2, \ldots . N-1$ and

$i = \sqrt{(-1)}$

There is another transformation that can also be used

Transformation 2 b (to detect − 1 symbol, C_0 equivalent pulse)

$d_i = i^{-b_i}$ for $i = 0, 1, 2, \ldots . N-1$

Transformation 3 (to detect +1 symbolC_1 equivalent pulse)

$y_i = (-1)^i$ for $i = 0, 1, 2, \ldots N-1$
Given code $\{c_0, c_1, \ldots c_{N-1}\}$ where N is the number of elements in the sequence.
$a_i = 1$ if $c_i = 1$; for $i = 0, \ldots N-1$
$a_i = -1$ if $c_i = 0$; for $i = 0, \ldots N-1$
$b_0 = a_0 - / + a_N$;
$b_i = b_{i-1} + a_i - a_{i-1}$ for $i = 1, 2, \ldots N-1$;
$d_i = y_i\, i^{b_i}$ for $i = 0, 1, 2, \ldots N-1$ and
$i = \sqrt{(-1)}$
There is another transformation that can also be used
Transformation 3 b (to detect 1 symbol, C_1 equivalent pulse)
$d_i = i^{-b_i}$ for $i = 0, 1, 2, \ldots N-1$ and

Transformation 4 (to detect -1 symbolC_1 equivalent pulse)
$y_i = (-1)^i$ for $i = 0, 1, 2, \ldots N-1$
Given code $\{c_0, c_1, \ldots c_{N-1}\}$ where N is the number of elements in the sequence.
$a_i = -1$ if $c_i = 1$; for $i = 0, \ldots N-1$
$a_i = 1$ if $c_i = 0$; for $i = 0, \ldots N-1$
$b_0 = a_0 - / + a_N$;
$b_i = b_{i-1} + a_i - a_{i-1}$ for $i = 1, 2, \ldots N-1$;
$d_i = y_i\, i^{b_i}$ for $i = 0, 1, 2, \ldots N-1$ and
$i = \sqrt{(-1)}$
There is another transformation that can also be used
Transformation 4 b (to detect -1 symbol, C_1 equivalent pulse)
$d_i = i^{-b_i}$ for $i = 0, 1, 2, \ldots N-1$ and

We also claim any
a)cyclic rotation of the sequences resulting from Transformations 1,2,3,4 and 1b,2b,3b,4b
b)multiplication of the elements of the sequence obtained from Transformations 1,2,3,4 and 1b,2b,3b,4b by a constant (real or complex)
are also claimed to be transformations that we wish to patent.
We also claim any combination of cyclic rotation followed by linear scaling of the result is patented.

The subroutine CodingTransformation implements the transformations 1,2,3,4

We also stress that while the rest of the bitstream is differentially encoded and the substitution used is $\{0 \to 1, 1 \to -1\}$ the substitution elsewhere used for the training sequences is $\{0 \to -1, 1 \to 1\}$

6)The demodulation in the CDMA mode uses the 4 separately transformed codes(Transformation 1,2,3,4) or (Transformation 1,2). With thus using 2 pulses and 4 at most complex correlations, we can demodulate the bitstream. Thus, most of the work is done by the gold code, during correlation.
7)There is nothing special about the Gold Code used in the example. Any gold like code will do as long as it posseses the usual properties of a gold code.

8) In case we consider a BT = 0.15 etc GSM modulation scheme, (i.e. twice the speed) we can keep the

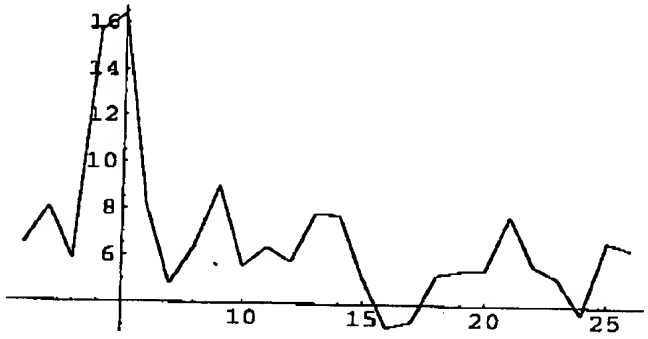
Example of the output using different encoders which encode transformations to correlate with the input of a bipolar bit stream of the first training sequence of GSM system where we have used the rule {0 –> –1} and not differentially encoded the training sequence .

The first training sequence of GSM system where we have used the rule {0 –> –1} and not differentially encoded it is calles GSMseq ={–1,–1,1,–1,–1,1,–1,1,1,1,–1,–1,–1,–1,1,–1,–1,1,–1,–1,1,–1,1,1,1,1}

```
ModOutputGSM = Modulator[L][ GSMseq, NumberOfCurves → 2,
  ModulatingPulse → OptPulseScaled, SamplingInterval → T / 4];
```

We call the Modulator output ModOutputGSM. PrimitiveCDMAReceiver encodes GSMseq according to transformation rule 1

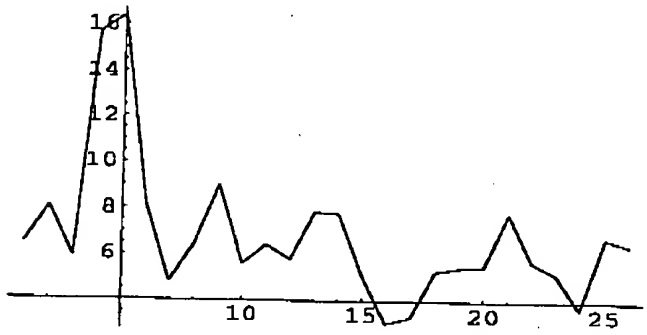
```
TomGSM1 = PrimitiveCDMAReceiver[ModOutputGSM, GSMseq, 1, 4];

ListPlot[TomGSM1 // Abs, PlotJoined -> True, PlotRange -> All]
```



- Graphics -

We have plotted the autocorrelation. The next graph shows the autocorrelation when GSMseq is encoded with transformation 1b. It seems the same as with transformation 1b

```
PrimitiveCDMAReceiver2[ModOutputGSM, GSMseq, 1, 4];

ListPlot[% // Abs, PlotJoined -> True, PlotRange -> All]
```

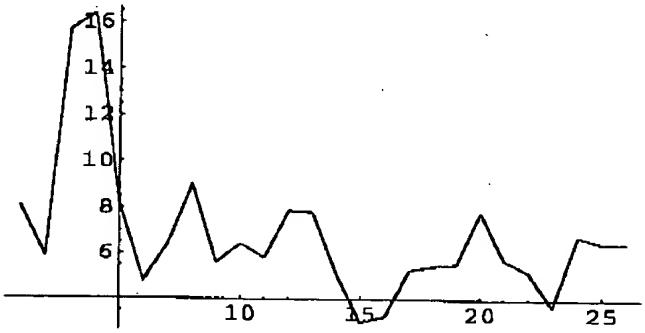


- Graphics -

PrimitiveCDMAReceiverGSM2Pulse encodes GSMseq with Transform 3

```
PrimitiveCDMAReceiverGSM2Pulse[ModOutputGSM, GSMseq, 1, 4];
```

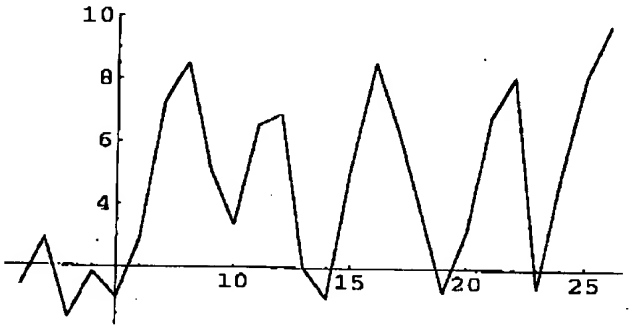
```
ListPlot[% // Abs, PlotJoined -> True, PlotRange -> All]
```




```
- Graphics -
```

PrimitiveCDMAReceiverGSM2PulseEfficient encodes GSMseq with Transform3b. This seems to be diffrent

```
PrimitiveCDMAReceiverGSM2PulseEfficient[ModOutputGSM, GSMseq, 1, 4];

ListPlot[% // Abs, PlotJoined -> True, PlotRange -> All]
```

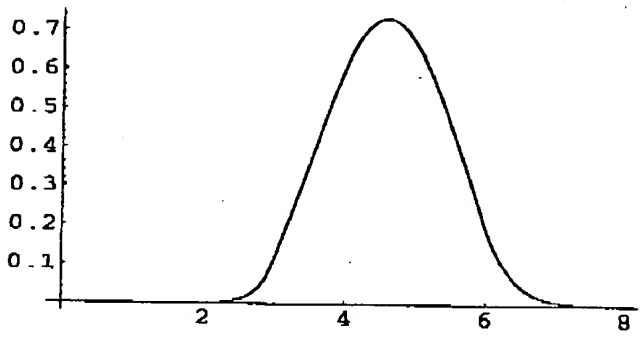



```
- Graphics -
```

```
Needs["RingFunctionsDiffEncoded`"]

Names["RingFunctionsDiffEncoded`*"]

{AllGoldSequences, AutocorrelationSequence, AutomorphismSigma, CodingTransform,
 CodingTransformNew, CrosscorrelationSequence, CyclicMultiplativeGroup,
 DropLeadingZeros, GoldSequence, GraeffeMethod, InitialConditions, MinimumPoly,
 ModuloMultplication, PolynomialMultiplication, PossibleDivisors, RingDivision,
 RingPower, SequenceGenerator, SpecifiedGoldSequences, TraceRepresentation,
 TupleRepresentation, TwoAdicExpansion, UnitsRing, ZeroPad, ZeroSequences, τ, σ}

Needs["LaurentFunctions`"]

RuleDelayed::rhs : Pattern t_ appears on the right-hand side of rule
   PhaseAngle[L_][t_] :> (PhaseAngle[L][t_] = Module[{x1, x2, x3, x4, x5, x6}, <<1>>]).

Needs["LaurentNotationTest`"]

Needs::nocont : Context LaurentNotationTest` was not created when Needs was evaluated.
```

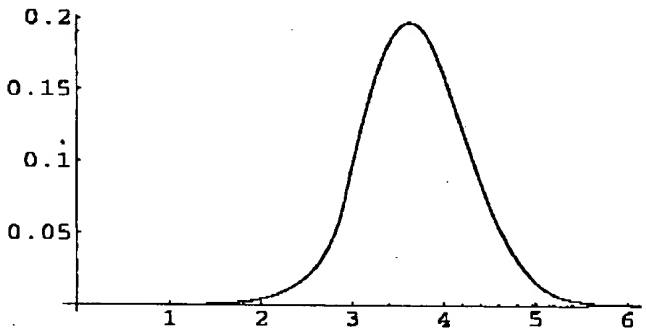
Information on the functions used can be obtained using help.

```
Names["LaurentFunctions`*"]

{AKN, AlphaKI, ANKInitialStateSetUp, BT, FiltPulse, h, hFiltered, InitialState, J,
 LaurentC, LaurentLK, LaurentS, M, ModulatingPulse, ModulationIndex, Modulator,
 NumberOfCurves, PhaseAngle, PhaseAngleFast, Receiver, ReceiverProper, S,
 SamplingInterval, StartingQuadrant, SyncSample, T, C, Φ, ψ}

T := 3/812500
BT := 0.3

ModulationIndex := 1/2

<< ModulatorData.m;

<< OptimalPulseShapes.m;

Plot[OptPulse[L][0][t], {t, 0, 8}]
```




```
- Graphics -

Table[
```

```
Plot[OptPulse[L][1][t], {t, 0 , 6}]
```



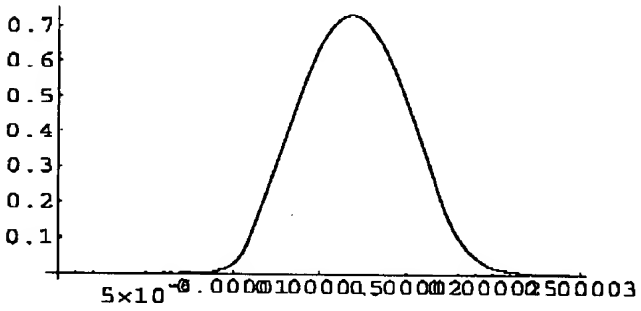

```
- Graphics -
```

```
T
```

$$\frac{3}{812500}$$

The unit of time is T = 1 for OptPulse. We scale the Pulses to $T = \frac{3}{812500}$ for the unit of time.

```
OptPulseScaled[8][0][t_] := OptPulse[L][0][t / T]
OptPulseScaled[8][1][t_] := OptPulse[L][1][t / T]

Plot[OptPulseScaled[L][0][t], {t, 0 , 8 T}]
```

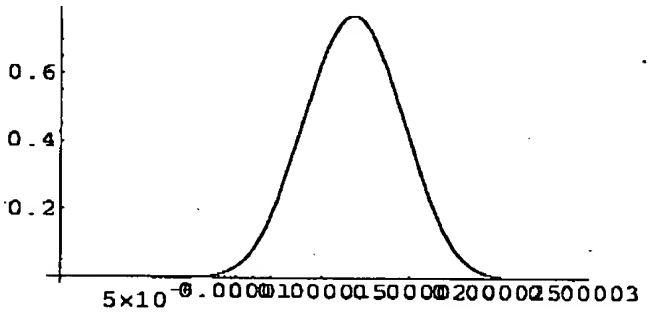



```
- Graphics -
```

```
RandomBitSeq
```

```
{1, 1, -1, -1, -1, 1, 1, -1, 1, -1, 1, -1, 1, -1, -1, -1, -1, 1, 1, 1, -1, -1, -1, 1, 1, 1,
 -1, 1, -1, -1, 1, -1, -1, 1, 1, 1, 1, -1, 1, -1, 1, -1, 1, -1, -1, 1, 1, 1, 1, -1,
 1, -1, 1, -1, 1, -1, 1, 1, -1, -1, 1, -1, 1, -1, 1, 1, 1, 1, -1, -1, -1, 1, -1,
 1, 1, 1, -1, 1, -1, -1, 1, 1, 1, -1, -1, 1, 1, 1, 1, 1, 1, 1, 1, 1, -1, -1, 1, -1, 1, -1}
```

```
Plot[FiltPulse[L][0][t], {t, 0, 8 T}]
```




```
- Graphics -
```

■ The Gold Sequence Set

We generate the sequences using the method specified by Serdar Boztas and P Vijay Kumar in Ref [1]. The numbering of the sequences is the one used in the paper. We generate a small subset of the sequences. There are $2^{10}+1$ sequences with the quaternary polynomial used. Given any binary primitive polynomial, we can generate the corresponding quaternary polynomial.

```
Goldseqlist =
 SpecifiedGoldSequences[{1, 3, 2, 1, 0, 3, 0, 0, 1}][{1, 2, 3, Last - 1, Last}];
```

The last sequence of seqlist has nice autocorelation properties.

The last sequence is in fact a m–sequence of length 1023 bits.

```
Goldseqlist // Last
```

```
{1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0,
 0, 1, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0,
 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 0,
 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 1, 0,
 0, 1, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0,
 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1,
 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1,
 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0,
 1, 1, 0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0}
```

```
AutocorrelationSequence[Goldseqlist // Last]
```

```
{255, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
 -1
```

```
AutocorrelationSequence[Goldseqlist.// #[[3]]&]
```

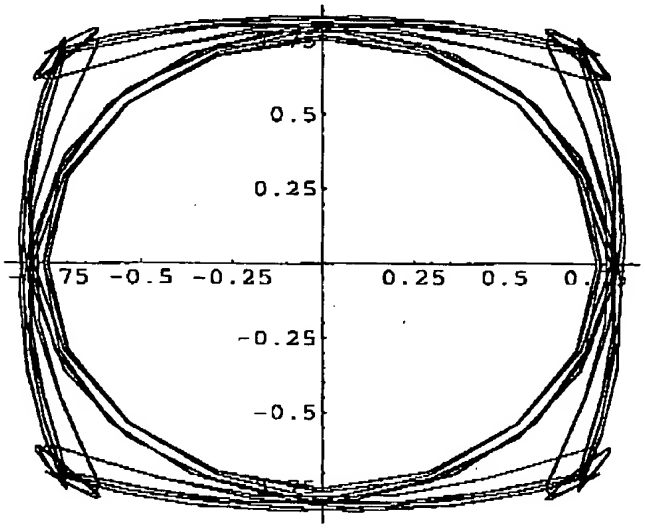
```
{255, -1, -1, -1, -1, -1, -1, -17, -1, -1, -1, -17, -1, -17, -17, -17, -1, -17, -1, -1,
 -1, -1, 15, -1, -1, 15, 15, -17, -17, -1, -17, 15, -1, -1, -17, 15, -1, 15, -1, 15, -1,
 15, -1, -1, 15, -17, -1, 15, -1, -1, -17, -17, -17, -1, -17, 15, 15, 15, -1, -1, 15,
 -17, 15, -1, -1, -1, -1, 15, 15, -1, 15, -1, -1, -17, -17, 15, -1, -1, 15, 15, -1, -1,
 -17, -1, -1, -17, -1, -1, 15, -1, 15, -17, -1, -1, -17, -1, -1, 15, -1, -17, 15, -1, 15,
 -1, 15, 15, -1, 15, -17, 15, 15, -1, 15, -1, 15, -17, -1, -1, -1, 15, -17, -17, 15, -1,
 -17, -1, -1, -1, -1, -1, -1, -17, -1, 15, -17, -17, 15, -1, -1, -1, -17, 15, -1, 15, -1,
 15, 15, -17, 15, -1, 15, 15, -1, 15, -1, 15, -17, -1, 15, -1, -1, -17, -1, -1, -17,
 15, -1, 15, -1, -1, -17, -1, -1, -17, -1, -1, 15, 15, -1, -1, 15, -17, -17,
 -1, -1, 15, -1, 15, 15, -1, -1, -1, -1, 15, -17, 15, -1, -1, 15, 15, 15, -17,
 -1, -17, -17, -17, -1, -1, 15, -1, -17, 15, -1, -1, 15, -1, 15, -1, 15, -1,
 15, -17, -1, -1, 15, -17, -1, -17, -17, 15, 15, -1, -1, 15, -1, -1, -1, -1,
 -17, -1, -17, -17, -17, -1, -17, -1, -1, -1, -17, -1, -1, -1, -1, -1, -1}
```

We generate the output of the modulator

```
ModOutput = Modulator[L][(Goldseqlist // Last) /. {0 -> -1}, NumberOfCurves → 2,
  ModulatingPulse → FiltPulse, SamplingInterval → T / 4];
```

We check the output

```
ListPlot[{Re[ModOutput], Im[ModOutput]} // Transpose, PlotJoined -> True,
 AspectRatio -> 1]
```



- Graphics -

Now we try to test the modulated sequence using the

The number of samples per chip is equal to

$$\frac{3}{26000000} \Big/ T$$

$$\frac{1}{32}$$

```
(Receiver[L][ModOutput, StartingQuadrant -> 0,
    SamplingInterval → T / 4, ModulatingPulse → FiltPulse] // Drop[#, 4]& ) -
 ( (Goldseqlist // Last // Drop[#, -4]&) /. { 0 -> -1})
```

```
{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 
```

```
CDMAEncode[BiPolarBitSeq_, GoldSeq_] :=
  Module[{x1, x2, x3},
  x1 = GoldSeq /. {0 → -1};
  Map[x1 # &, BiPolarBitSeq] // Flatten]

CDMAEncodedSeq = CDMAEncode[{-1, 1, 1, -1}, Goldseqlist // Last];
```

▪ CDMA decoding of single Symbol

The modular output associated with {1}

```
ModOutputPlusOne = Modulator[L][CDMAEncode[{1}, (Goldseqlist // Last) /. {0 -> -1}],
  NumberOfCurves → 2, ModulatingPulse → OptPulseScaled, SamplingInterval → T / 4];
```

This is a primitive decoder built to study the autocorrelation. This will help in decoding

```
Take[ModOutputPlusOne, 10]
```

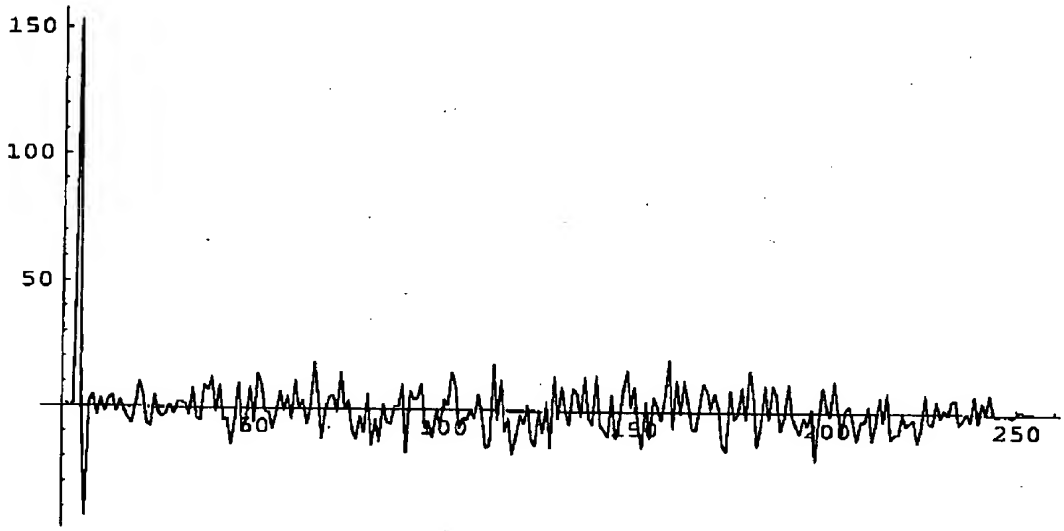
```
{0.691379 + 0.510132 I, 0.431257 + 0.748432 I, 0.130196 + 0.863809 I,
 -0.183585 + 0.853405 I, -0.510127 + 0.69136 I, -0.748423 + 0.431231 I,
 -0.863782 + 0.130145 I, -0.853326 - 0.183682 I, -0.69115 - 0.510321 I,
 -0.430757 - 0.74887 I}
```

```
PrimitiveCDMAReceiver[ModOutput_, GoldSeq_, Sample_, OverSampling_] :=
 Module[{x1, x2State, x3Update, x4, x5State, x6},
  x1 = Partition[ModOutput, OverSampling] // Transpose // #[[Sample]]&;
  x2State = GoldSeq /. {0 -> -1};
   x5State = Table[(-1)^Mod[i, 2], {i, 0, Length[x2State] - 1}];
  x3Update := Module[{},
    x2State = RotateRight[x2State]; x4 = FoldList[Plus, 0, x2State] // Rest // I^#&;
   x5State = RotateRight[x5State]; x1 (x5State x4) // Apply[Plus, #]&];
Table[x3Update, {i, 1, Length[GoldSeq]}]]
```

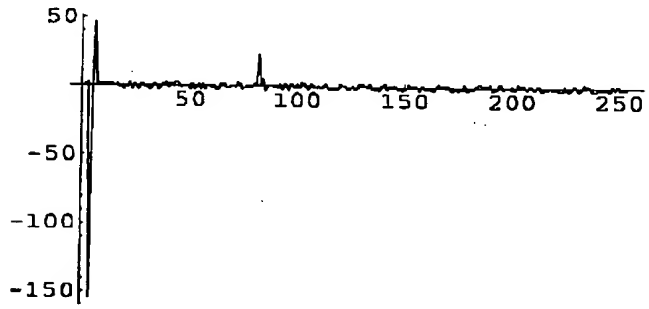
We make it more efficient

```
PrimitiveCDMAReceiver2[ModOutput_, GoldSeq_, Sample_, OverSampling_] :=
 Module[{x1, x2State, x3Update, x4, x5State, x6},
  x1 = Partition[ModOutput, OverSampling] // Transpose // #[[Sample]]&;
  x2State = GoldSeq /. {0 -> -1};
   x5State = Table[(1)^Mod[i, 2], {i, 0, Length[x2State] - 1}];
  x3Update := Module[{},
    x2State = RotateRight[x2State]; x4 = FoldList[Plus, 0, x2State] // Rest // I^-#&;
   x5State = RotateRight[x5State]; x1 (x5State x4) // Apply[Plus, #]&];
Table[x3Update, {i, 1, Length[GoldSeq]}]]

Tom = PrimitiveCDMAReceiver[ModOutputPlusOne, (Goldseqlist // Last), 1, 4];
```

```
ListPlot[Tom // Re, PlotJoined -> True, PlotRange -> All]
```

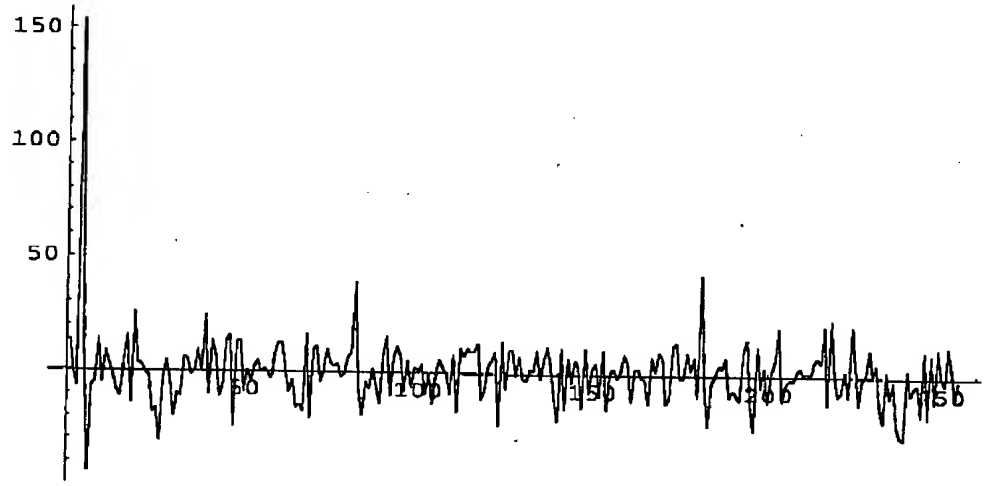



```
- Graphics -

Tom = PrimitiveCDMAReceiver2[ModOutputPlusOne, (Goldseqlist // Last), 1, 4];

ListPlot[Tom // Re, PlotJoined -> True, PlotRange -> All]
```

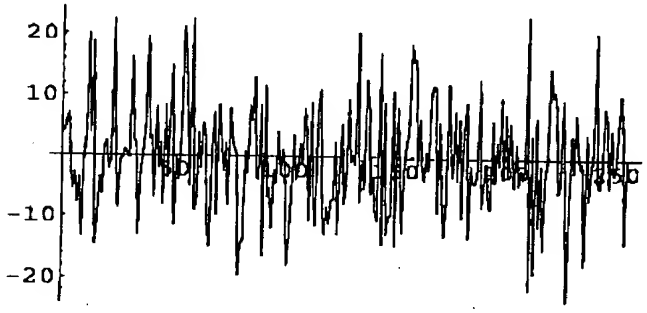



```
- Graphics -

Tom3 = PrimitiveCDMAReceiver2[ModOutputPlusOne3, (Goldseqlist // #[[3]]&), 1, 4];
```

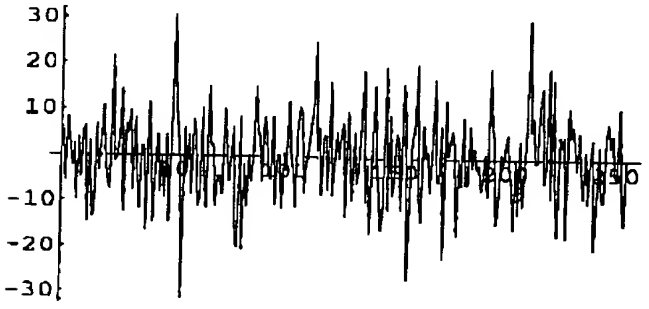
```
ListPlot[Tom3 // Re, PlotJoined -> True, PlotRange -> All]
```




```
- Graphics -

Tom4 = PrimitiveCDMAReceiver2[ModOutputPlusOne3, (Goldseqlist // #[[4]]&), 1, 4];
ListPlot[Tom4 // Re, PlotJoined -> True, PlotRange -> All]
```




```
- Graphics -

Tom5 = PrimitiveCDMAReceiver[ModOutputPlusOne3, (Goldseqlist // #[[4]]&), 1, 4];
ListPlot[Tom5 // Re, PlotJoined -> True, PlotRange -> All]
```




```
- Graphics -
```

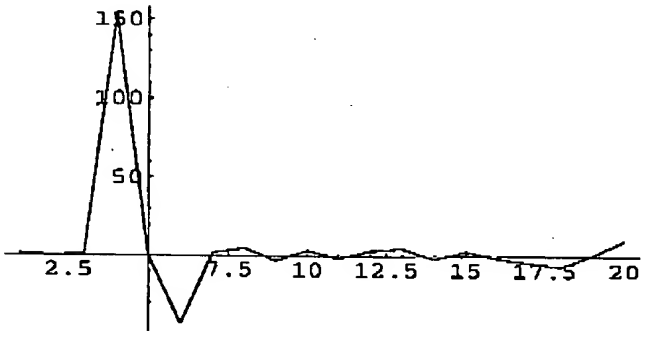
```
Length[Tom]
```

255

```
Take [Tom, 20]
```

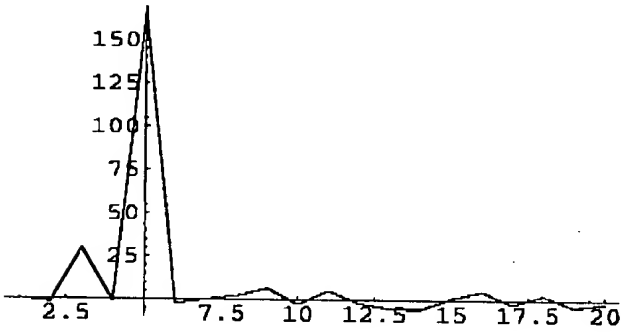
```
{1.75501 + 0.329995 I, 0.0343421 - 1.58366 I, 1.39349 + 29.6902 I, 153.017 - 0.957068 I,
 -0.718377 + 165.563 I, -43.6424 - 2.04777 I, 2.21031 + 0.256636 I, 5.392 + 2.18642 I,
 -3.13017 + 6.54167 I, 3.61193 - 2.5452 I, -1.68586 + 5.24864 I, 3.35077 - 1.96193 I,
 4.93408 - 5.10993 I, -1.90672 - 5.29083 I, 3.34843 + 0.914983 I, -1.24192 + 4.93512 I,
 -3.90572 - 2.50768 I, -6.70085 + 2.60649 I, 0.886488 - 4.00636 I, 10.5807 - 2.08322 I}
```

```
ListPlot[Tom // Re // Take[#, 20]&, PlotJoined -> True, PlotRange -> All]
```



- Graphics -

```
ListPlot[Tom // Im // Take[#, 20]&, PlotJoined -> True, PlotRange -> All]
```

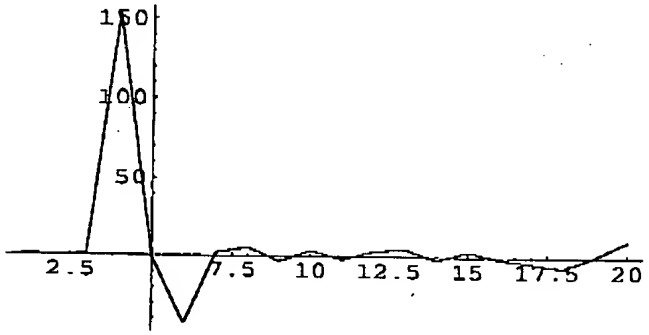


- Graphics -

```
Take[Tom, 10]
```

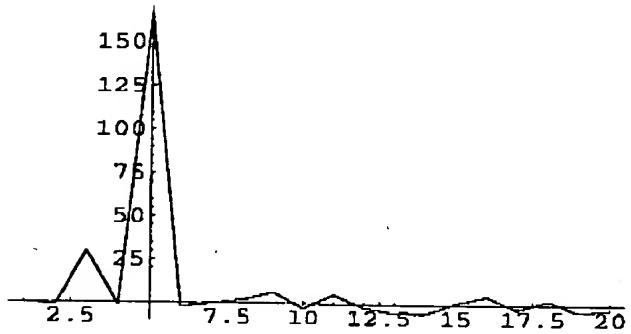
```
{1.75501 + 0.329995 I, 0.0343421 - 1.58366 I, 1.39349 + 29.6902 I, 153.017 - 0.957068 I,
 -0.718377 + 165.563 I, -43.6424 - 2.04777 I, 2.21031 + 0.256636 I, 5.392 + 2.18642 I,
 -3.13017 + 6.54167 I, 3.61193 - 2.5452 I}
```

```
ListPlot[Tom // Re // Take[#, 20]&, PlotJoined -> True, PlotRange -> All]
```



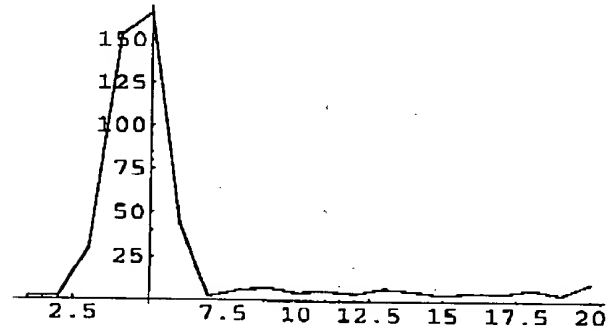

```
- Graphics -
```

```
ListPlot[Tom // Im // Take[#, 20]&, PlotJoined -> True, PlotRange -> All]
```




```
- Graphics -
```

```
ListPlot[Tom // Abs // Take[#, 20]&, PlotJoined -> True, PlotRange -> All]
```

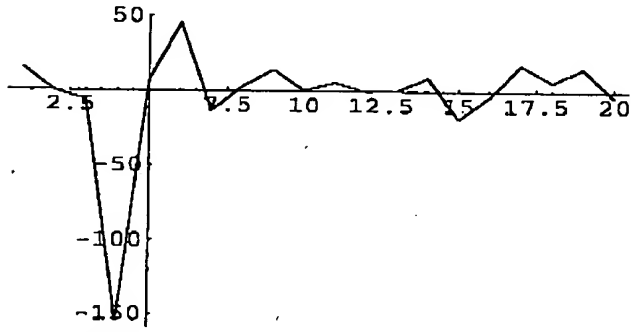



```
- Graphics -
```

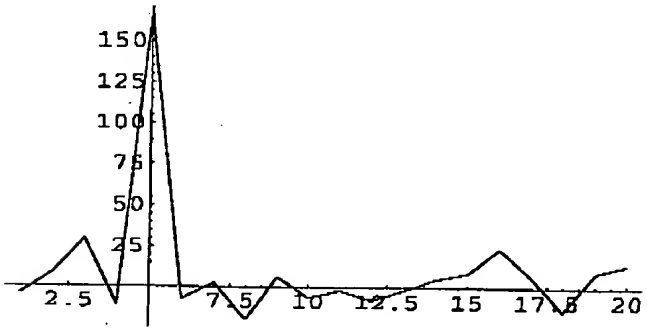
We try a less favourable sequence

```
ModOutputPlusOne3 = Modulator[L][CDMAEncode[{1}, Goldseqlist // #[[3]]&],
  NumberOfCurves → 2, ModulatingPulse → OptPulseScaled, SamplingInterval → T / 4];
```

```
PrimitiveCDMAReceiverMinus[ModOutput_, GoldSeq_, Sample_, OverSampling_] :=
 Module[{x1, x2State, x3Update, x4, x5State, x6},
  x1 = Partition[ModOutput, OverSampling] // Transpose // #[[Sample]]&;
  x2State = -(GoldSeq /. {0 -> -1});
   x5State = Table[(-1)^Mod[i, 2], {i, 0, Length[x2State] - 1}];
  x3Update := Module[{},
    x2State = RotateRight[x2State]; x4 = FoldList[Plus, 0, x2State] // Rest // I^#&;
   x5State = RotateRight[x5State]; x1 (x5State x4) // Apply[Plus, #]&];
Table[x3Update, {i, 1, Length[GoldSeq]}]]

Tom4 = PrimitiveCDMAReceiver[ModOutputPlusOne3, (Goldseqlist // #[[3]]&), 1, 4];

Take[Tom4, 10]

{14.7675 - 4.24542 I, -0.301874 + 8.61808 I, -5.92939 + 29.6038 I, -154.61 - 10.4754 I,
 6.64138 + 166.152 I, 45.2887 - 7.16233 I, -12.809 + 2.47796 I, 3.45948 - 19.7653 I,
 14.2097 + 6.33624 I, 0.585872 - 6.10244 I}

ListPlot[Tom4 // Re // Take[#, 20]&, PlotJoined -> True, PlotRange -> All]
```

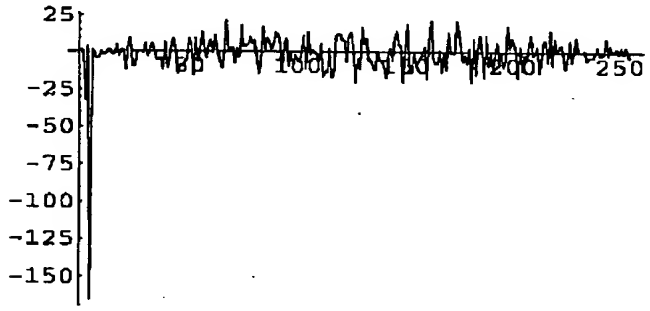



```
- Graphics -

ListPlot[Tom4 // Im // Take[#, 20]&, PlotJoined -> True, PlotRange -> All]
```




```
- Graphics -

ModOutputMinusOne =
 Modulator[L][CDMAEncode[{-1}, (Goldseqlist // Last) /. {0 -> -1}],
  NumberOfCurves → 2, ModulatingPulse → OptPulseScaled, SamplingInterval → T / 4];

TomM1 = PrimitiveCDMAReceiverMinus[ModOutputMinusOne, (Goldseqlist // Last), 1, 4];
```

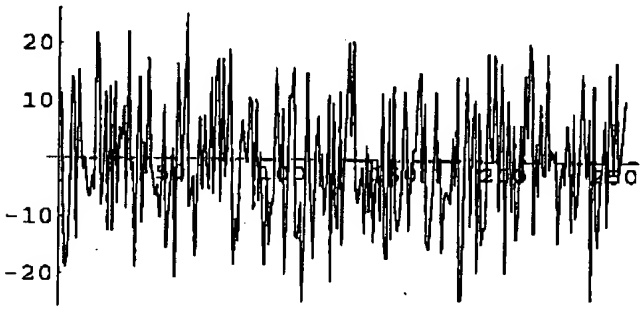
```
ListPlot[TomM1 // Im, PlotJoined -> True, PlotRange -> All]
```




```
- Graphics -
```

```
TomM1 = PrimitiveCDMAReceiver[ModOutputMinusOne, (Goldseqlist // Last), 1, 4];
```

```
ListPlot[TomM1 // Im, PlotJoined -> True, PlotRange -> All]
```




```
- Graphics -
```

```
Length[Tom]
```

```
255
```

```
Take [Tom, 20]
```

```
{1.75501 + 0.329995 I, 0.0343421 - 1.58366 I, 1.39349 + 29.6902 I, 153.017 - 0.957068 I,
 -0.718377 + 165.563 I, -43.6424 - 2.04777 I, 2.21031 + 0.256636 I, 5.392 + 2.18642 I,
 -3.13017 + 6.54167 I, 3.61193 - 2.5452 I, -1.68586 + 5.24864 I, 3.35077 - 1.96193 I,
 4.93408 - 5.10993 I, -1.90672 - 5.29083 I, 3.34843 + 0.914983 I, -1.24192 + 4.93512 I,
 -3.90572 - 2.50768 I, -6.70085 + 2.60649 I, 0.886488 - 4.00636 I, 10.5807 - 2.08322 I}
```

Checking the correlation properties of the Training Sequence in GSM

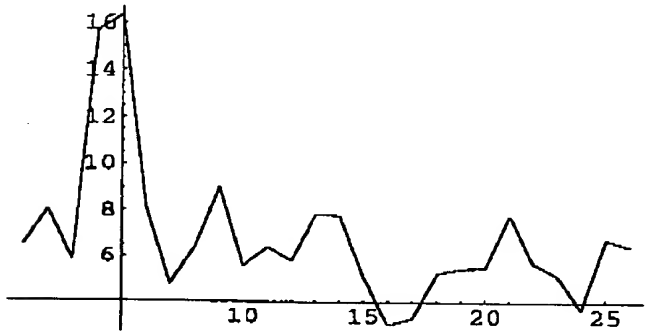
```
GSM = {0, 0, 1, 0, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 1}
```

```
{0, 0, 1, 0, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 1}
```

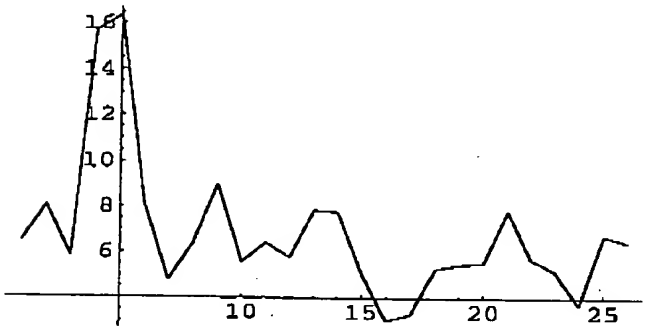
```
GSMseq = {-1, -1, 1, -1, -1, 1, -1, 1, 1, 1, -1, -1, -1, -1, 1, -1, -1, -1, 1,
  -1, -1, 1, -1, 1, 1, 1}
```

```
{-1, -1, 1, -1, -1, 1, -1, 1, 1, 1, -1, -1, -1, -1, 1, -1, -1, -1, 1, -1, -1,
 1, -1, 1, 1, 1}
```

```
AutocorrelationSequence[GSMseq]

{26, -2, -2, 2, -2, -2, -2, 6, -10, 2, 10, -2, -2, -2, -2, -2, 10, 2, -10, 6,
 -2, -2, -2, 2, -2, -2}

ModOutputGSM = Modulator[L][GSMseq, NumberOfCurves → 2,
  ModulatingPulse → OptPulseScaled, SamplingInterval → T / 4];

TomGSM1 = PrimitiveCDMAReceiver[ModOutputGSM, GSMseq, 1, 4];

ListPlot[TomGSM1 // Abs, PlotJoined -> True, PlotRange -> All]
```

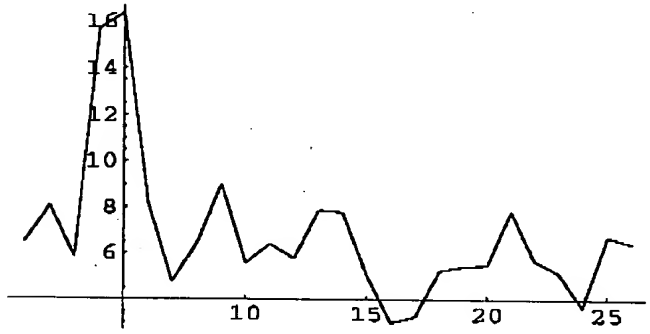



```
- Graphics -

PrimitiveCDMAReceiver2[ModOutputGSM, GSMseq, 1, 4];

ListPlot[%90 // Abs, PlotJoined -> True, PlotRange -> All]
```



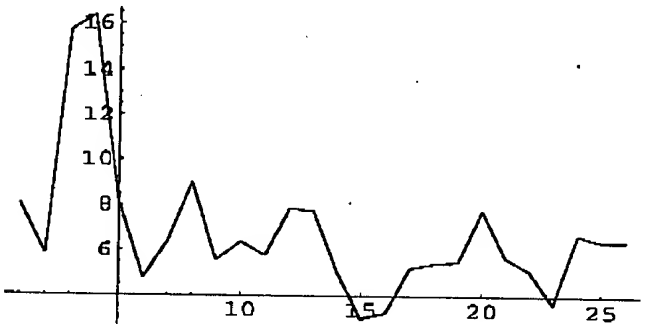

```
- Graphics -

PrimitiveCDMAReceiver[ModOutputGSM, GSMseq, 1, 4];
```

```
ListPlot[%92 // Abs, PlotJoined -> True, PlotRange -> All]
```



- Graphics -

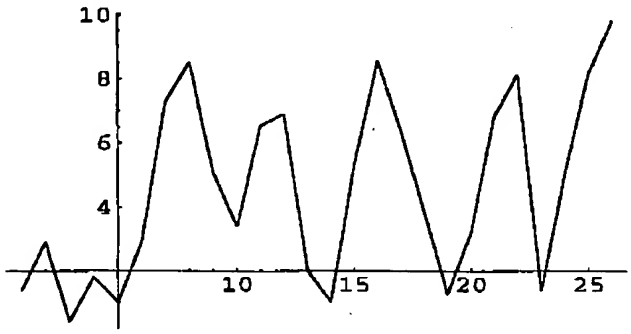
```
PrimitiveCDMAReceiverGSM2Pulse[ModOutput_, GoldSeq_, Sample_, OverSampling_] :=
 Module[{x1, x2State, x3Update, x4, x5State, x5, x6},
  x1 = Partition[ModOutput, OverSampling] // Transpose // #[[Sample]]&;
  x2State = GoldSeq /. {0 -> -1};
   x5State = Table[(-1)^Mod[i, 2], {i, 0, Length[x2State] - 1}];
  x3Update := Module[{}, x2State = RotateRight[x2State];
    x4 = FoldList[Plus, 0, x2State] // Rest;
      x5 = Join[{1}, x2State // Drop[#, -1]&];
      x6 = FoldList[Plus, 0, x5] // Rest // I^#&;
   x5State = RotateRight[x5State]; x1 (x5State x6) // Apply[Plus, #]&];
Table[x3Update, {i, 1, Length[GoldSeq]}]]

TomGSMSecondP = PrimitiveCDMAReceiverGSM2Pulse[ModOutputGSM, GSMseq, 1, 4];

ListPlot[TomGSMSecondP // Abs, PlotJoined -> True, PlotRange -> All]
```



- Graphics -

```
PrimitiveCDMAReceiverGSM2PulseEfficient[
  ModOutput_, GoldSeq_, Sample_, OverSampling_] :=
 Module[{x1, x2State, x3Update, x4, x5State, x5, x6},
  x1 = Partition[ModOutput, OverSampling] // Transpose // #[[Sample]]&;
  x2State = GoldSeq /. {0 -> -1};
   x5State = Table[(1)^Mod[i, 2], {i, 0, Length[x2State] - 1}];
  x3Update := Module[{}, x2State = RotateRight[x2State];
    x4 = FoldList[Plus, 0, x2State] // Rest;
      x5 = Join[{1}, x2State // Drop[#, -1]&];
      x6 = FoldList[Plus, 0, x5] // Rest // I^#&;
   x5State = RotateRight[x5State]; x1 (x5State x6) // Apply[Plus, #]&];
Table[x3Update, {i, 1, Length[GoldSeq]}]]

TomGSMSecondPEff2 =
 PrimitiveCDMAReceiverGSM2PulseEfficient[ModOutputGSM, GSMseq, 1, 4];
```

```
ListPlot[TomGSMSecondPEff2 // Abs, PlotJoined -> True, PlotRange -> All]
```




```
- Graphics -
```

▪ CDMA Decoding of Several Symbols with Training sequence Receiver

First we generate the modulator output. For simplicity we will use a very short training sequence . Let the training sequence one of GSM training sequences Let the guard sequences be {1,1,1} . Let the data symbols be generated by a random

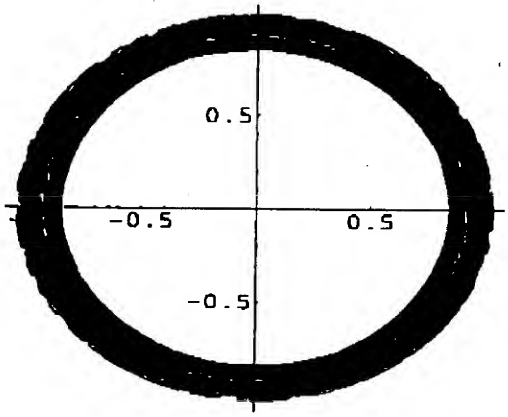
```
data1 = {1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 0} ;

data2 = {0, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1};

guard = {1, 1, 1}

{1, 1, 1}

training = {0, 0, 1, 0, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 1};
```

The GSM Training sequence is {0,0,1,0,0,1,0,1,1,1,0,0,0,0,1,0,0,0,1,0,0,1,0,1,1,1}. In fact any short m—sequence can be used to characterise the output.

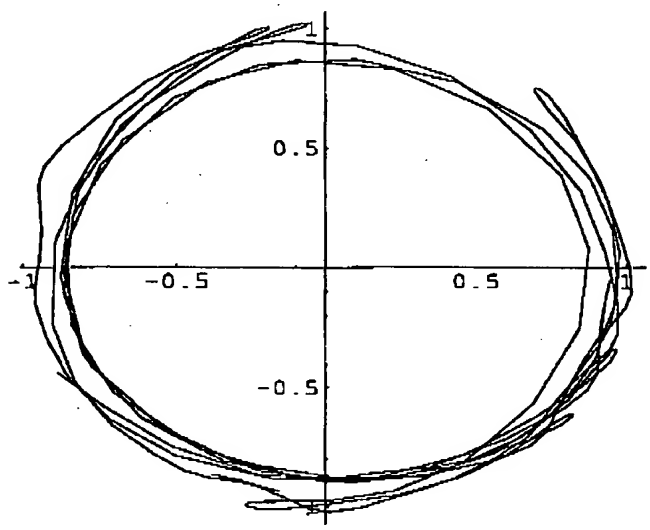
Only at this point that we differentially encode using the gsm scheme

```
frame = Join[guard, data1, training, data2, guard]

General::spell1 :
 Possible spelling error: new symbol name "frame" is similar to existing symbol "Frame".

{1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0,
 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 1, 0, 0, 1, 0, 1,
 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1}

GSMDiffEncodedFrame[frame_] := Module[{x1, x2, x3}, x1 = Partition[frame, 2, 1];
   x2 = Map[Mod[#[[1]] + #[[2]], 2]&, x1] // Join[{frame[[1]]}, #]&;
  x3 = x2 /. {0 -> 1, 1 -> -1}]

General::spell1 :
 Possible spelling error: new symbol name "frame" is similar to existing symbol "Frame".

frameEncoded = GSMDiffEncodedFrame[frame]

{-1, 1, 1, 1, 1, 1, 1, 1, 1, -1, -1, -1, -1, -1, -1, 1, 1, -1, -1, 1, -1, -1, -1, 1, 1,
 -1, -1, 1, -1, -1, -1, -1, 1, 1, -1, 1, 1, 1, -1, -1, 1, 1, 1, -1, -1, 1, -1, -1, -1,
 1, 1, -1, 1, -1, -1, -1, 1, -1, -1, -1, -1, 1, -1, -1, 1, -1, -1, -1, 1, -1, 1, 1, 1, 1}

ModOutputFrame3 = Modulator[L][CDMAEncode[frameEncoded, Goldseqlist // Last],
   NumberOfCurves → 2, ModulatingPulse → OptPulseScaled, SamplingInterval → T/4];
```

```
Carrfreq001 = Table[E^(I 2 Pi 0.001 j) // N, {j, 0, Length[ModOutputFrame3] - 1}];

Carr001 = ModOutputFrame3 Carrfreq001;

Save["Carr001.m", Carr001];

Carrfreq0005 = Table[E^(I 2 Pi 0.0005 j) // N, {j, 0, Length[ModOutputFrame3] - 1}];

Carr0005 = ModOutputFrame3 Carrfreq0005;

Save["Carr0005.m", Carr0005];

Carrfreq002 = Table[E^(I 2 Pi 0.002 j) // N, {j, 0, Length[ModOutputFrame3] - 1}];

Carr002 = ModOutputFrame3 Carrfreq002;

Save["Carr002.m", Carr002];

ModOutputUnEncodedFrame3 =
 Modulator[L][CDMAEncode[frame /. 0 -> -1, Goldseqlist // #[[3]]&],
  NumberOfCurves → 2, ModulatingPulse → OptPulseScaled, SamplingInterval → T / 4];

Save["ModOutputUnEncodedFrameGSMLike3.m", ModOutputFrame]

<< ModOutputFrameEncodedGSMLike.m;

Length[ModOutputFrame]

73440

AFC0005 = Take[Carr0005, {50, 7300}];

AFC0005 // {Re[#], Im[#]}& // Transpose //
 ListPlot[#, PlotJoined -> True, PlotRange -> All, AspectRatio -> 1]&;
```



```
AFC0005 // Take[#, 200]& // {Re[#], Im[#]}& // Transpose //
 ListPlot[#, PlotJoined -> True, PlotRange -> All, AspectRatio -> 1]&;
```



In the PrimitiveCDMA receiver we need to specify the sample. In the CDMA synchroniser we discover the sample.

```
CDMACoarseSynchroniserNew[ModOutput_, GoldSeq_, Threshold_, OverSampling_] :=
 Module[{x1, x2, x3, x4Plus, x4Minus, x5State,
   x6Count, x6MaxCorr, seq, x7Update, x8, x9, x10, x11, x12, x13},
  x1 = ModOutput;
  x2 = GoldSeq /. 0 -> -1;
  x3 = CodingTransformNew[L][GoldSeq];
  x4Plus = x3[[1]];
  x4Minus = x3[[2]];
  x5State = Take[x1, (Length[x2] + 1) OverSampling];
  x6Count = 1;
  x6MaxCorr = 0;
  seq = Drop[x1, OverSampling (Length[x2] + 1)];
 x7Update := Module[{},
x8 = Partition[x5State, OverSampling] // Transpose;
x9 = Map[{Drop[#, -1] . x4Plus, Drop[#, 1] . x4Minus}&, x8];
x10 = Map[
      {Abs[Im[#[[1]]]], Abs[Re[#[[1]]]] . Abs[Re[#[[2]]]] , Abs[Im[#[[2]]]] }&, x9];
x11 = If[Max[x10] > Threshold, Throw[{x10, x6Count, True}],
      {x6Count, x10, x6MaxCorr = Max[x6MaxCorr, x10], False}];
    x6Count = x6Count + 1;
    x5State = Join[Drop[x5State, OverSampling], Take[seq, OverSampling]];
    seq = Drop[seq, OverSampling];
    x11 ];
x12 = Catch[Table[x7Update, {i, 1, Length[seq] / OverSampling }]];
 x13 = If[Last[x12] === True,
    CDMAFineSynchroniser[ModOutput, x12[[2]], x3, OverSampling],
    {"Failed to Coarse Synchronise", False}]]

Tom4 =
 CDMACoarseSynchroniserNew[AFC0005 // Drop[#, 250 4]&, Goldseqlist // Last , 50, 4]

DeModulator[
 {{{16.252 - 103.911 I, 9.19889 + 6.93038 I, -3.61972 + 15.2763 I, 13.0046 - 9.41072 I},
   {-102.686 - 22.7446 I, 7.49431 - 8.74558 I, 15.0189 + 4.57178 I, -8.57558 - 13.5698 I},
   {-29.1474 + 101.055 I, -8.25775 - 8.02866 I, 5.50581 - 14.7022 I, -14.0815 + 7.70661 I},
   {99.0255 + 35.4352 I, -8.53133 + 7.73733 I, -14.3275 - 6.4181 I, 6.80721 + 14.5376 I},
   {41.5832 - 96.6051 I, 7.18638 + 9.00033 I, -7.30507 + 13.8962 I, 14.9364 - 5.88096 I}},
  {{24.6533 - 96.8492 I, 10.1266 + 6.85283 I, 1.46802 - 2.37691 I, 12.5087 - 10.488 I},
   {-95.1101 - 30.6858 I, 7.47516 - 9.6763 I,
    -2.28004 - 1.61437 I, -9.68188 - 13.1425 I}, {-36.5973 + 92.9956 I,
    -9.18784 - 8.06799 I, -1.75435 + 2.17418 I, -13.7245 + 8.83755 I},
   {90.5141 + 42.3643 I, -8.62898 + 8.66312 I, 2.05973 + 1.8874 I, 7.95834 + 14.2523 I},
   {47.9641 - 87.6755 I, 8.1042 + 9.15591 I, 2.01301 - 1.93715 I, 14.7239 - 7.04772 I}}}]
```

```
Tom0005 =
 CDMACoarseSynchroniserNew[Carr0005 // Drop[#, 250 4]&, Goldseqlist // Last , 100, 4];

CDMAFineSynchroniser[ModOutput_,
  ThresholdCorrelatioCount_, CorrelatingSeq_, OverSampling_] :=
   Module[{x1, x2, x3, x4, x5, x6, seq, x7Update, x8First,
   x8Second, x9First, x9Second, x10First, x10Second, x11, x12, x13, x14},
  x1 = If[ThresholdCorrelatioCount > 3,
    ThresholdCorrelatioCount - 3, ThresholdCorrelatioCount];
  x2 = CorrelatingSeq;
  x3 = Drop[ModOutput, x1 OverSampling];
 x4 =  CDMAPositionFinder[x3, x2, OverSampling];
  x5 = CDMAPositionFinder[Drop[x3, Length[x2] OverSampling] , x2, OverSampling];
   x6 = CDMAPositionFinder[Drop[x3, 2 Length[x2] OverSampling] , x2, OverSampling];
  x7 = PositionAverager[{x4[[1]], x5[[1]], x6[[1]]}];
  x8First = Drop[x3,  (x7[[1]] - 1) OverSampling] //
      Partition[#, OverSampling]& // Transpose // #[[x7[[2]]]]&;
   x8Second = Drop[x3, x7[[1]] OverSampling] //
      Partition[#, OverSampling]& // Transpose // #[[x7[[2]]]]&;
 x9First = x8First // Partition[#, Length[x2[[1]]]]&;
  x10First = Map[Function[x, Map[x . #&, x2]], x9First];
    x9Second = x8Second // Partition[#, Length[x2[[1]]]]&;
  x10Second = Map[Function[x, Map[x . #&, x2]], x9Second] ;
  DeModulator[{x10First, x10Second}]   ]

CDMAPositionFinder[ModOutput_, CorrelatingSeq_, OverSampling_] :=
    Module[{x5State, x6Count, seq, x7Update, x8, x9, x10, x11, x12, x13},
 x5State = Take[ModOutput, ( Length[CorrelatingSeq[[1]]] + 1) OverSampling];
  x6Count = 1;
  seq = Drop[ModOutput, OverSampling ( Length[CorrelatingSeq[[1]]] + 1)];
  x7Update := Module[{},
x8 = Partition[x5State, OverSampling] // Transpose;
x9 =  Map[{Drop[#, -1] . CorrelatingSeq[[1]], Drop[#, -1] . CorrelatingSeq[[2]],
        Drop[#, 1] . CorrelatingSeq[[1]], Drop[#, 1] . CorrelatingSeq[[2]]}&, x8];
    x6Count = x6Count + 1;
    x5State = Join[Drop[x5State, OverSampling], Take[seq , OverSampling]];
    seq = Drop[seq, OverSampling];
    x9 ];
x10 =  Table[x7Update, {i, 1, 10}];
x11 =  MapIndexed[Max[{Abs[Re[#]], Abs[Im[#]]}]&, x10, {3}];
x12 = MapIndexed[Apply[Plus, #]&, x11, {2}];
  x13 = Position[x12, Max[x12]] ]
```

We need to define a position averager. We for now just take the first element

```
PositionAverager[PositionList_] := First[PositionList]

Tom4 =
 CDMACoarseSynchroniserNew[TestData // Drop[#, 250 4]&, Goldseqlist // Last , 100, 4]

DeModulator[
 {{{-157.056 + 0.691379 I, 7.37139 - 18.8305 I, 41.1322 + 1.94693 I, -16.2843 - 23.4715 I},
   {-0.691379 - 157.056 I, 18.8305 + 7.37139 I, -1.94693 + 41.1322 I, 23.4715 - 16.2843 I},
   {157.056 - 0.691379 I, -7.37139 + 18.8305 I, -41.1322 - 1.94693 I, 16.2843 + 23.4715 I},
   {0.691379 + 157.056 I, -18.8305 - 7.37139 I,
    1.94693 - 41.1322 I, -23.4715 + 16.2843 I}, {-157.056 + 0.691379 I,
    7.37139 - 18.8305 I, 41.1322 + 1.94693 I, -16.2843 - 23.4715 I}},
  {{-169.734 - 0.510132 I, 6.8871 - 20.5026 I, 6.24607 + 1.47947 I, -17.4944 - 21.5101 I},
   {0.510132 - 169.734 I, 20.5026 + 6.8871 I, -1.47947 + 6.24607 I, 21.5101 - 17.4944 I},
   {169.734 + 0.510132 I, -6.8871 + 20.5026 I, -6.24607 - 1.47947 I, 17.4944 + 21.5101 I},
   {-0.510132 + 169.734 I, -20.5026 - 6.8871 I, 1.47947 - 6.24607 I, -21.5101 + 17.4944 I},
   {-169.734 - 0.510132 I, 6.8871 - 20.5026 I, 6.24607 + 1.47947 I,
    -17.4944 - 21.5101 I}}}]
```

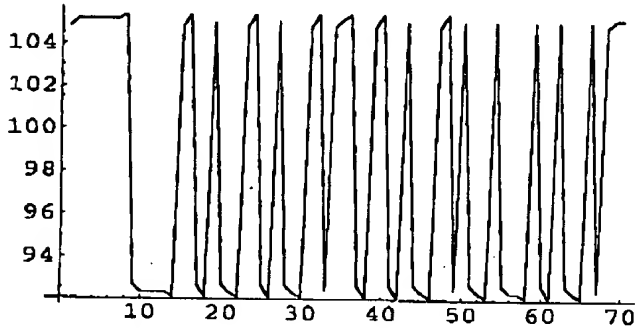
```
Tom4[[1]] // Transpose
```

```
{{{-157.056 + 0.691379 I, 7.37139 - 18.8305 I, 41.1322 + 1.94693 I, -16.2843 - 23.4715 I},
   {-169.734 - 0.510132 I, 6.8871 - 20.5026 I, 6.24607 + 1.47947 I, -17.4944 - 21.5101 I}},
  {{-0.691379 - 157.056 I, 18.8305 + 7.37139 I, -1.94693 + 41.1322 I, 23.4715 - 16.2843 I},
   {0.510132 - 169.734 I, 20.5026 + 6.8871 I, -1.47947 + 6.24607 I, 21.5101 - 17.4944 I}},
  {{157.056 - 0.691379 I, -7.37139 + 18.8305 I, -41.1322 - 1.94693 I, 16.2843 + 23.4715 I},
   {169.734 + 0.510132 I, -6.8871 + 20.5026 I, -6.24607 - 1.47947 I, 17.4944 + 21.5101 I}},
  {{0.691379 + 157.056 I, -18.8305 - 7.37139 I, 1.94693 - 41.1322 I, -23.4715 + 16.2843 I},
   {-0.510132 + 169.734 I, -20.5026 - 6.8871 I, 1.47947 - 6.24607 I, -21.5101 + 17.4944 I}},
  {{-157.056 + 0.691379 I, 7.37139 - 18.8305 I, 41.1322 + 1.94693 I, -16.2843 - 23.4715 I},
   {-169.734 - 0.510132 I, 6.8871 - 20.5026 I, 6.24607 + 1.47947 I, -17.4944 - 21.5101 I}}}
```

```
Take[Tom5[[1]] // Transpose, {8, 10}]
```

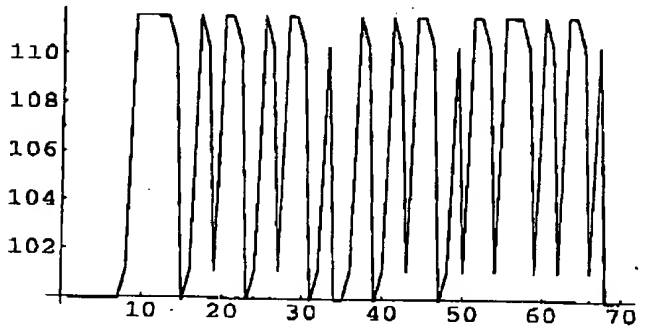
```
{{{157.25 - 0.929033 I, -7.57128 + 19.0518 I, -40.9109 - 1.74704 I, 16.522 + 23.6653 I},
   {169.813 - 0.714545 I, -7.42576 + 21.3328 I, -5.4158 - 0.940778 I, 18.7191 + 21.5891 I}},
  {{19.1272 - 6.93509 I, -1.02745 + 156.682 I, 23.9098 + 15.9863 I, -1.57069 - 41.4696 I},
   {20.5326 - 6.8667 I, 0.478141 + 169.715 I, 21.5305 + 17.4644 I, -1.46045 - 6.27806 I}},
  {{-7.37139 - 18.8305 I, 157.056 + 0.691379 I, 16.2843 - 23.4715 I, -41.1322 + 1.94693 I},
   {-6.8871 - 20.5026 I, 169.734 - 0.510132 I, 17.4944 - 21.5101 I, -6.24607 + 1.47947 I}}}
```

```
Tom6[[1, 1]] // Transpose // {#[[1]], #[[2]]}& // Transpose // Map[Max, Abs[#]]& //
 ListPlot[#, PlotJoined -> True]&
```



- Graphics -

```
Tom6[[1, 2]] // Transpose // {#[[1]], #[[2]]}& // Transpose // Map[Max, Abs[#]]& //
 ListPlot[#, PlotJoined -> True]&
```



- Graphics -

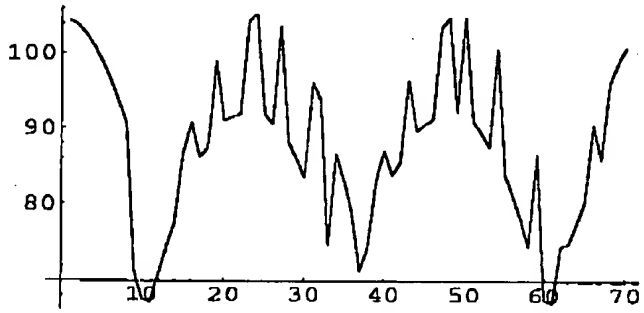
```
Map[Abs, Tom6[[1, 2]]]
```

```
{{99.9069, 12.1968, 2.82297, 16.3585}, {99.9377, 12.2274, 2.7937, 16.3237},
 {99.9377, 12.2274, 2.7937, 16.3237}, {99.9377, 12.2274, 2.7937, 16.3237},
 {99.9377, 12.2274, 2.7937, 16.3237}, {99.9377, 12.2274, 2.7937, 16.3237},
 {99.9377, 12.2274, 2.7937, 16.3237}, {101.16, 12.0496, 2.71211, 17.0171},
 {10.9504, 111.653, 17.2325, 16.2024}, {10.9205, 111.621, 17.2666, 16.1692},
 {10.9205, 111.621, 17.2666, 16.1692}, {10.9205, 111.621, 17.2666, 16.1692},
 {10.9205, 111.621, 17.2666, 16.1692}, {11.1111, 110.39, 16.592, 16.2327},
 {99.9069, 12.1968, 2.82297, 16.3585}, {101.16, 12.0496, 2.71211, 17.0171},
 {10.9504, 111.653, 17.2325, 16.2024}, {11.1111, 110.39, 16.592, 16.2327},
 {101.129, 12.0176, 2.74759, 17.0524}, {10.9504, 111.653, 17.2325, 16.2024},
 {10.9205, 111.621, 17.2666, 16.1692}, {11.1111, 110.39, 16.592, 16.2327},
 {99.9069, 12.1968, 2.82297, 16.3585}, {101.16, 12.0496, 2.71211, 17.0171},
 {10.9504, 111.653, 17.2325, 16.2024}, {11.1111, 110.39, 16.592, 16.2327},
 {101.129, 12.0176, 2.74759, 17.0524}, {10.9504, 111.653, 17.2325, 16.2024},
 {10.9205, 111.621, 17.2666, 16.1692}, {11.1111, 110.39, 16.592, 16.2327},
 {99.9069, 12.1968, 2.82297, 16.3585}, {101.16, 12.0496, 2.71211, 17.0171},
 {11.1392, 110.422, 16.5587, 16.2668}, {99.9069, 12.1968, 2.82297, 16.3585},
 {99.9377, 12.2274, 2.7937, 16.3237}, {101.16, 12.0496, 2.71211, 17.0171},
 {10.9504, 111.653, 17.2325, 16.2024}, {11.1111, 110.39, 16.592, 16.2327},
 {99.9069, 12.1968, 2.82297, 16.3585}, {101.16, 12.0496, 2.71211, 17.0171},
 {10.9504, 111.653, 17.2325, 16.2024}, {11.1111, 110.39, 16.592, 16.2327},
 {101.129, 12.0176, 2.74759, 17.0524}, {10.9504, 111.653, 17.2325, 16.2024},
 {10.9205, 111.621, 17.2666, 16.1692}, {11.1111, 110.39, 16.592, 16.2327},
 {99.9069, 12.1968, 2.82297, 16.3585}, {101.16, 12.0496, 2.71211, 17.0171},
 {11.1392, 110.422, 16.5587, 16.2668}, {101.129, 12.0176, 2.74759, 17.0524},
 {10.9504, 111.653, 17.2325, 16.2024}, {10.9205, 111.621, 17.2666, 16.1692},
 {11.1111, 110.39, 16.592, 16.2327}, {101.129, 12.0176, 2.74759, 17.0524},
 {10.9504, 111.653, 17.2325, 16.2024}, {10.9205, 111.621, 17.2666, 16.1692},
 {10.9205, 111.621, 17.2666, 16.1692}, {11.1111, 110.39, 16.592, 16.2327},
 {101.129, 12.0176, 2.74759, 17.0524}, {10.9504, 111.653, 17.2325, 16.2024},
 {11.1111, 110.39, 16.592, 16.2327}, {101.129, 12.0176, 2.74759, 17.0524},
 {10.9504, 111.653, 17.2325, 16.2024}, {10.9205, 111.621, 17.2666, 16.1692},
 {11.1111, 110.39, 16.592, 16.2327}, {101.129, 12.0176, 2.74759, 17.0524},
 {11.1392, 110.422, 16.5587, 16.2668}, {99.9069, 12.1968, 2.82297, 16.3585},
 {99.9377, 12.2274, 2.7937, 16.3237}, {99.9377, 12.2274, 2.7937, 16.3237}}
```

```
Max0005 =
 Map[{Max[{Re[#[[1]]] // Abs, Im[#[[1]]] // Abs, Re[#[[2]]] // Abs, Im[#[[2]]] // Abs}],
    Max[{Re[#[[3]]] // Abs, Im[#[[3]]] // Abs, Re[#[[4]]] // Abs, Im[#[[4]]] // Abs}]}&,
  Tom6[[1, 1]]]
```

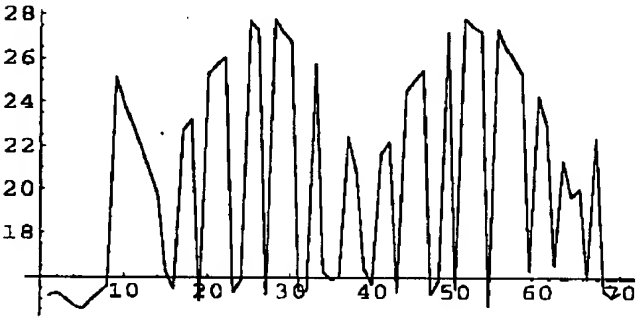
```
{{104.351, 15.1379}, {103.911, 15.2763}, {102.686, 15.0189},
 {101.055, 14.7022}, {99.0255, 14.5376}, {96.6051, 14.9364},
 {93.8035, 15.2762}, {90.7175, 15.6415}, {71.4359, 25.1654}, {67.5608, 23.8519},
 {67.0266, 22.9468}, {70.8801, 21.9511}, {74.4539, 20.8688}, {77.6444, 19.7766},
 {86.7967, 16.2927}, {90.7238, 15.4739}, {86.1537, 22.7114}, {87.5152, 23.2037},
 {98.9714, 14.9013}, {91.0278, 25.3506}, {91.5367, 25.7331}, {91.8871, 26.0709},
 {104.382, 15.2963}, {105.346, 15.8922}, {91.9518, 27.7219}, {90.4855, 27.2999},
 {103.704, 15.2097}, {88.1543, 27.846}, {85.9746, 27.2075}, {83.3983, 26.8136},
 {96.1447, 15.3897}, {93.9076, 15.3803}, {74.6965, 25.8353}, {86.612, 16.2707},
 {83.2288, 15.9297}, {79.0562, 16.0518}, {71.3823, 22.4556}, {74.383, 20.923},
 {82.9341, 16.4519}, {87.1763, 15.7147}, {83.839, 21.6473}, {85.5445, 22.25},
 {96.5637, 15.3294}, {89.7552, 24.5667}, {90.6053, 25.0753}, {91.3309, 25.5023},
 {103.567, 15.2284}, {104.931, 15.891}, {92.2522, 27.2697}, {104.992, 15.4722},
 {91.0436, 27.8733}, {89.5104, 27.4849}, {87.6329, 27.2718}, {100.78, 14.7073},
 {83.8748, 27.3795}, {81.0829, 26.501}, {78.152, 25.9901}, {74.6068, 25.3375},
 {86.6792, 16.3378}, {67.5876, 24.357}, {66.9937, 22.9638}, {74.5498, 16.5852},
 {74.9528, 21.37}, {77.7339, 19.7041}, {80.5993, 20.0831}, {90.616, 16.0026},
 {86.0108, 22.4409}, {96.2752, 15.4329}, {98.8878, 15.1306}, {100.942, 15.3636}}
```

```
ListPlot[Max0005 // Transpose // #[[1]]&, PlotJoined -> True]
```




```
- Graphics -
```

```
ListPlot[Max0005 // Transpose // #[[2]]&, PlotJoined -> True]
```




```
- Graphics -
```

```
Map[{Max[{Re[#[[1]]] // Abs, Im[#[[1]]] // Abs, Re[#[[2]]] // Abs, Im[#[[2]]] // Abs}],
   Max[{Re[#[[3]]] // Abs, Im[#[[3]]] // Abs, Re[#[[4]]] // Abs, Im[#[[4]]] // Abs}]}&,
 Tom5[[1, 2]]]
```

```
{{169.715, 21.5305}, {169.734, 21.5101}, {169.734, 21.5101},
 {169.734, 21.5101}, {169.734, 21.5101}, {169.734, 21.5101},
 {169.734, 21.5101}, {169.813, 21.5891}, {169.715, 21.5305}, {169.734, 21.5101},
 {169.734, 21.5101}, {169.734, 21.5101}, {169.734, 21.5101}, {169.813, 21.5891},
 {169.715, 21.5305}, {169.813, 21.5891}, {169.715, 21.5305}, {169.813, 21.5891},
 {169.794, 21.6095}, {169.715, 21.5305}, {169.734, 21.5101}, {169.813, 21.5891},
 {169.715, 21.5305}, {169.813, 21.5891}, {169.715, 21.5305}, {169.813, 21.5891},
 {169.794, 21.6095}, {169.715, 21.5305}, {169.734, 21.5101}, {169.813, 21.5891},
 {169.715, 21.5305}, {169.813, 21.5891}, {169.794, 21.6095}, {169.715, 21.5305},
 {169.734, 21.5101}, {169.813, 21.5891}, {169.715, 21.5305}, {169.813, 21.5891},
 {169.715, 21.5305}, {169.813, 21.5891}, {169.715, 21.5305}, {169.813, 21.5891},
 {169.794, 21.6095}, {169.715, 21.5305}, {169.734, 21.5101}, {169.813, 21.5891},
 {169.715, 21.5305}, {169.813, 21.5891}, {169.794, 21.6095}, {169.794, 21.6095},
 {169.715, 21.5305}, {169.734, 21.5101}, {169.813, 21.5891}, {169.794, 21.6095},
 {169.715, 21.5305}, {169.734, 21.5101}, {169.734, 21.5101}, {169.813, 21.5891},
 {169.794, 21.6095}, {169.715, 21.5305}, {169.813, 21.5891}, {169.794, 21.6095},
 {169.715, 21.5305}, {169.734, 21.5101}, {169.813, 21.5891}, {169.794, 21.6095},
 {169.794, 21.6095}, {169.715, 21.5305}, {169.734, 21.5101}, {169.734, 21.5101}}
```

```
demodframe = Sign[Re[ Tom5[[1, 1]] SeqI]] /. {-1 -> 1, 1 -> 0}
```

```
{1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0,
 1, 0, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 1, 0, 0, 1, 0, 1, 1,
 0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1}
```

The first and last bits have been lost in the processing

```
frame

{1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0,
 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 1, 0, 0, 1, 0, 1,
 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1}

Length[frame]

72

truncatedframe = frame // Drop[#, 1]& // Drop[#, -1]&

{1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0,
 1, 0, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 1, 0, 0, 1, 0, 1, 1,
 0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1}

demodframe - truncatedframe

{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}

Tom6 = CDMACoarseSynchroniser[ModOutputUnEncodedFrame3 // Drop[#, 250 4]&,
  Goldseqlist // #[[3]]& , 100, 4]

$Aborted

Tom6[[1, 1]] SeqI // Re

{-157.37, -157.056, -157.056, -157.056, -157.056, -157.056, -157.056, -156.862,
 9.75774, -9.3815, 9.3815, -9.3815, 9.3815, -9.18122, -157.37, -156.862, 9.75774,
 -9.18122, -157.176, 9.75774, -9.3815, 9.18122, 157.37, 156.862, -9.75774, 9.18122,
 157.176, -9.75774, 9.3815, -9.18122, -157.37, -156.862, 9.55746, 157.37, 157.056,
 156.862, -9.75774, 9.18122, 157.37, 156.862, -9.75774, 9.18122, 157.176, -9.75774,
 9.3815, -9.18122, -157.37, -156.862, 9.55746, 157.176, -9.75774, 9.3815, -9.18122,
 -157.176, 9.75774, -9.3815, 9.3815, -9.18122, -157.176, 9.75774, -9.18122,
 -157.176, 9.75774, -9.3815, 9.18122, 157.176, -9.55746, -157.37, -157.056, -157.056}

Tom6[[1, 1]] SeqI // Im

{-10.0454, -10.3815, -10.3815, -10.3815, -10.3815, -10.3815, -10.3815, -10.62,
 -15.3789, 15.6783, -15.6783, 15.6783, -15.6783, 15.9004, -10.0454, -10.62, -15.3789,
 15.9004, -10.2839, -15.3789, 15.6783, -15.9004, 10.0454, 10.62, 15.3789, -15.9004,
 10.2839, 15.3789, -15.6783, 15.9004, -10.0454, -10.62, -15.601, 10.0454, 10.3815,
 10.62, 15.3789, -15.9004, 10.0454, 10.62, 15.3789, -15.9004, 10.2839,
 15.3789, -15.6783, 15.9004, -10.0454, -10.62, -15.601, 10.2839, 15.3789,
 -15.6783, 15.9004, -10.2839, -15.3789, 15.6783, -15.6783, 15.9004, -10.2839,
 -15.3789, 15.9004, -10.2839, -15.3789, 15.6783, -15.9004, 10.2839, 15.601,
 -10.0454, -10.3815, -10.3815}

Tom6[[1, 2]] SeqI // Re

{-169.751, -169.733, -169.733, -169.733, -169.733, -169.733, -169.733, -169.638,
 10.444, -10.425, 10.425, -10.425, 10.425, -9.86944, -169.751, -169.638, 10.444,
 -9.86944, -169.656, 10.444, -10.425, 9.86944, 169.751, 169.638, -10.444, 9.86944,
 169.656, -10.444, 10.425, -9.86944, -169.751, -169.638, 9.88847, 169.751, 169.733,
 169.638, -10.444, 9.86944, 169.751, 169.638, -10.444, 9.86944, 169.656, -10.444,
 10.425, -9.86944, -169.751, -169.638, 9.88847, 169.656, -10.444, 10.425, -9.86944,
 -169.656, 10.444, -10.425, 10.425, -9.86944, -169.656, 10.444, -9.86944,
 -169.656, 10.444, -10.425, 9.86944, 169.656, -9.88847, -169.751, -169.733, -169.733}
```

```
Tom6 = CDMACoarseSynchroniser[ModOutputUnEncodedFrame3 // Drop[#, 250 4]&,
  Goldseqlist // #[[3]]& , 100, 4]

DeModulator[{{10.3815 - 157.056 I, 157.056 + 10.3815 I, -10.3815 + 157.056 I,
   -157.056 - 10.3815 I, 10.3815 - 157.056 I, 157.056 + 10.3815 I, -10.3815 + 157.056 I,
   -156.862 - 10.62 I, 15.601 + 9.55746 I, -157.176 - 10.2839 I, 15.601 + 9.55746 I,
   -157.176 - 10.2839 I, 15.601 + 9.55746 I, -157.37 - 10.0454 I, 10.3815 - 157.056 I,
   156.862 + 10.62 I, -15.601 - 9.55746 I, 157.37 + 10.0454 I, -10.62 + 156.862 I,
   9.55746 - 15.601 I, -10.2839 + 157.176 I, 9.75774 - 15.3789 I, -15.6783 - 9.3815 I,
   -9.18122 + 15.9004 I, 10.2839 - 157.176 I, -9.75774 + 15.3789 I, 15.9004 + 9.18122 I,
   -157.176 - 10.2839 I, 15.601 + 9.55746 I, -157.37 - 10.0454 I, 10.3815 - 157.056 I,
   156.862 + 10.62 I, -15.3789 - 9.75774 I, -9.3815 + 15.6783 I, 15.6783 + 9.3815 I,
   9.18122 - 15.9004 I, -10.2839 + 157.176 I, 9.75774 - 15.3789 I, -15.6783 - 9.3815 I,
   -9.18122 + 15.9004 I, 10.2839 - 157.176 I, -9.75774 + 15.3789 I, 15.9004 + 9.18122 I,
   -157.176 - 10.2839 I, 15.601 + 9.55746 I, -157.37 - 10.0454 I, 10.3815 - 157.056 I,
   156.862 + 10.62 I, -15.3789 - 9.75774 I, -9.18122 + 15.9004 I, 10.2839 - 157.176 I,
   -9.55746 + 15.601 I, 10.0454 - 157.37 I, 156.862 + 10.62 I, -15.601 - 9.55746 I,
   157.176 + 10.2839 I, -15.601 - 9.55746 I, 157.37 + 10.0454 I, -10.62 + 156.862 I,
   9.55746 - 15.601 I, -10.0454 + 157.37 I, -156.862 - 10.62 I, 15.601 + 9.55746 I,
   -157.176 - 10.2839 I, 15.3789 + 9.75774 I, 9.18122 - 15.9004 I, -10.0454 + 157.37 I,
   -157.056 - 10.3815 I, 10.3815 - 157.056 I, 157.056 + 10.3815 I}, {-7.65342 - 169.733 I,
   169.733 - 7.65342 I, 7.65342 + 169.733 I, -169.733 + 7.65342 I, -7.65342 - 169.733 I,
   169.733 - 7.65342 I, 7.65342 + 169.733 I, -169.638 + 6.41574 I, 17.2252 + 9.88847 I,
   -169.656 + 6.44773 I, 17.2252 + 9.88847 I, -169.656 + 6.44773 I, 17.2252 + 9.88847 I,
   -169.751 + 7.68541 I, -7.65342 - 169.733 I, 169.638 - 6.41574 I, -17.2252 - 9.88847 I,
   169.751 - 7.68541 I, 6.41574 + 169.638 I, 9.88847 - 17.2252 I, 6.44773 + 169.656 I,
   10.444 - 16.382 I, -16.412 - 10.425 I, -9.86944 + 17.2553 I, -6.44773 - 169.656 I,
   -10.444 + 16.382 I, 17.2553 + 9.86944 I, -169.656 + 6.44773 I, 17.2252 + 9.88847 I,
   -169.751 + 7.68541 I, -7.65342 - 169.733 I, 169.638 - 6.41574 I, -16.382 - 10.444 I,
   -10.425 + 16.412 I, 16.412 + 10.425 I, 9.86944 - 17.2553 I, 6.44773 + 169.656 I,
   10.444 - 16.382 I, -16.412 - 10.425 I, -9.86944 + 17.2553 I, -6.44773 - 169.656 I,
   -10.444 + 16.382 I, 17.2553 + 9.86944 I, -169.656 + 6.44773 I, 17.2252 + 9.88847 I,
   -169.751 + 7.68541 I, -7.65342 - 169.733 I, 169.638 - 6.41574 I, -16.382 - 10.444 I,
   -9.86944 + 17.2553 I, -6.44773 - 169.656 I, -9.88847 + 17.2252 I,
   -7.68541 - 169.751 I, 169.638 - 6.41574 I, -17.2252 - 9.88847 I,
   169.656 - 6.44773 I, -17.2252 - 9.88847 I, 169.751 - 7.68541 I, 6.41574 + 169.638 I,
   9.88847 - 17.2252 I, 7.68541 + 169.751 I, -169.638 + 6.41574 I, 17.2252 + 9.88847 I,
   -169.656 + 6.44773 I, 16.382 + 10.444 I, 9.86944 - 17.2553 I, 7.68541 + 169.751 I,
   -169.733 + 7.65342 I, -7.65342 - 169.733 I, 169.733 - 7.65342 I}}]

Tom6[[1, 1]] SeqI // Re

{157.056, 157.056, 157.056, 157.056, 157.056, 157.056, 157.056, 156.862, -9.55746,
 -157.176, 9.55746, 157.176, -9.55746, -157.37, -157.056, -156.862, 9.55746, 157.37,
 156.862, -9.55746, -157.176, 9.75774, -9.3815, 9.18122, 157.176, -9.75774, 9.18122,
 157.176, -9.55746, -157.37, -157.056, -156.862, 9.75774, -9.3815, 9.3815, -9.18122,
 -157.176, 9.75774, -9.3815, 9.18122, 157.176, -9.75774, 9.18122, 157.176, -9.55746,
 -157.37, -157.056, -156.862, 9.75774, -9.18122, -157.176, 9.55746, 157.37,
 156.862, -9.55746, -157.176, 9.55746, 157.37, 156.862, -9.55746, -157.37,
 -156.862, 9.55746, 157.176, -9.75774, 9.18122, 157.37, 157.056, 157.056, 157.056}

Tom6[[1, 1]] SeqI // Im

{-10.0454, -10.3815, -10.3815, -10.3815, -10.3815, -10.3815, -10.3815, -10.62,
 -15.3789, 15.6783, -15.6783, 15.6783, -15.6783, 15.9004, -10.0454, -10.62, -15.3789,
 15.9004, -10.2839, -15.3789, 15.6783, -15.9004, 10.0454, 10.62, 15.3789, -15.9004,
 10.2839, 15.3789, -15.6783, 15.9004, -10.0454, -10.62, -15.601, 10.0454, 10.3815,
 10.62, 15.3789, -15.9004, 10.0454, 10.62, 15.3789, -15.9004, 10.2839,
 15.3789, -15.6783, 15.9004, -10.0454, -10.62, -15.601, 10.2839, 15.3789,
 -15.6783, 15.9004, -10.2839, -15.3789, 15.6783, -15.6783, 15.9004, -10.2839,
 -15.3789, 15.9004, -10.2839, -15.3789, 15.6783, -15.9004, 10.2839, 15.601,
 -10.0454, -10.3815, -10.3815}

Tom6[[1, 2]] SeqI // Re

{-169.751, -169.733, -169.733, -169.733, -169.733, -169.733, -169.733, -169.638,
 10.444, -10.425, 10.425, -10.425, 10.425, -9.86944, -169.751, -169.638, 10.444,
 -9.86944, -169.656, 10.444, -10.425, 9.86944, 169.751, 169.638, -10.444, 9.86944,
 169.656, -10.444, 10.425, -9.86944, -169.751, -169.638, 9.88847, 169.751, 169.733,
 169.638, -10.444, 9.86944, 169.751, 169.638, -10.444, 9.86944, 169.656, -10.444,
 10.425, -9.86944, -169.751, -169.638, 9.88847, 169.656, -10.444, 10.425, -9.86944,
 -169.656, 10.444, -10.425, 10.425, -9.86944, -169.656, 10.444, -9.86944,
 -169.656, 10.444, -10.425, 9.86944, 169.656, -9.88847, -169.751, -169.733, -169.733}
```

```
Table[CDMAPositionFinder[ModOutputUnEncodedFrame3 // Drop[#, 250 4 + 1 20]&,
      Goldseqlist // #[[3]]&, 4] // Flatten // Abs // Max,
 {i, 251, 350}]
```

```
{22.6392, 22.6392, 22.6392, 17.4452, 17.6558, 17.6558, 17.6558, 17.6558, 17.4676,
 17.4676, 18.088, 18.088, 18.088, 18.088, 19.6342, 19.6342, 19.6342, 19.6342, 19.6342,
 13.457, 16.0791, 16.0791, 16.0791, 16.0791, 19.95, 19.95, 19.95, 19.95, 19.95,
 18.3042, 16.6993, 16.7542, 23.0831, 25.3773, 25.3773, 25.3773, 25.3773, 24.601,
 24.601, 24.601, 18.7221, 20.5255, 26.9209, 26.9209, 26.9209, 26.9209, 18.9705,
 18.9705, 18.9705, 20.3921, 22.6392, 22.6392, 22.6392, 22.6392, 17.4452, 17.6558,
 17.6558, 17.6558, 17.6558, 17.4676, 17.4676, 18.088, 18.088, 18.088, 18.088,
 19.6342, 19.6342, 19.6342, 19.6342, 19.6342, 13.457, 16.0791, 16.0791,
 16.0791, 16.0791, 19.95, 19.95, 19.95, 19.95, 19.95, 18.3042, 16.6993,
 16.7542, 23.0831, 25.3773, 25.3773, 25.3773, 25.3773, 24.601, 24.601, 24.601,
 18.7221, 20.5255, 26.9209, 26.9209, 26.9209, 26.9209, 18.9705, 18.9705, 18.9705}
```

```
Max[%]
```

```
26.9209
```

```
Map[Max, %]
```

```
{17.6558, 17.6558, 17.6558, 17.6558, 17.4676, 17.4676, 18.088, 18.088, 18.088, 18.088,
 19.6342, 19.6342, 19.6342, 19.6342, 19.6342, 13.457, 16.0791, 16.0791,
 16.0791, 16.0791, 19.95, 19.95, 19.95, 19.95, 19.95, 18.3042, 16.6993,
 16.7542, 23.0831, 25.3773, 25.3773, 25.3773, 25.3773, 24.601, 24.601, 24.601,
 18.7221, 20.5255, 26.9209, 26.9209, 26.9209, 26.9209, 18.9705, 18.9705,
 18.9705, 20.3921, 22.6392, 22.6392, 22.6392, 22.6392, 17.4452}
```

References

1 "Binary Sequences with Gold-Like Correlation but Larger Linear Span" by Serdar Boztas and P. Vijay Kumar IEEE Trans. on Information Theory Vol 40 No 2, March 1994

```
BeginPackage["RingFunctionsDiffEncoded`"]

RingDivision::usage = "RingDivision[ModuloRing][{Denominator,Numerator}] =
   {Quotitent,Remanider,Denominator,Flag} e.g If num = {1,0,0,0,0,0,0,0,0,0,1,
   0,0,0,0,0,1}.den = {1,0,0,1,0,1}. then RingDivision[2][{den,num}] = {{1,
   0,0,1,0,1,1,0,0,0,1},{1,0,1,0,0},{1,0,0,1,0,1},True}. When the
   algorithm encounters a zero devisor the flag is set to False. e.g RingDivision[
   4][{{2,1,1,1},{1,0,0,1}}] = {{},{1,0,0,1},{2,1,1,1},False} "

DropLeadingZeros::usage =
 "DropLeadingZeros[DenominatorSeq] drops the leading zeros in the sequence.
   e.g DropLeadingZeros[{3,3,1}] = {3,3,1}.  DropLeadingZeros[{0,0,0}] = {}"

PossibleDivisors::usage = "PossibleDivisors[ModuloRing][Order] gives the list of
   possible devisors. Note the MS power of the poly is LHS. PossibleDivisors[4][
   2] = {{1},{2},{3},{1,0},{1,1},{1,2},{1,3},{2,0},{2,1},{2,2},{2,3},{3,0},
   {3,1},{3,2},{3,3},{1,0,0},{1,0,1},{1,0,2},{1,0,3},{1,1,0},{1,1,1},{1,
   1,2},{1,1,3},{1,2,0},{1,2,1},{1,2,2},{1,2,3},{1,3,0},{1,3,1},{1,3,2},
   {1,3,3},{2,0,0},{2,0,1},{2,0,2},{2,0,3},{2,1,0},{2,1,1},{2,1,2},{2,1,
   3},{2,2,0},{2,2,1},{2,2,2},{2,2,3},{2,3,0},{2,3,1},{2,3,2},{2,3,3},
   {3,0,0},{3,0,1},{3,0,2},{3,0,3},{3,1,0},{3,1,1},{3,1,2},{3,1,
   3},{3,2,0},{3,2,1},{3,2,2},{3,2,3},{3,3,0},{3,3,1},{3,3,2},{3,3,3}}"

PolynomialMultiplication::usage = "PolynomialMultiplication[ModuloRing][
   poly1,poly2] performs polynomial multiplication moduloRing e.g.
   PolynomialMultiplication[4][{2,1},{2,1}] = {1}"

ModuloMultplication::usage =
 "ModuloMultplication[ModuloRing][PrimitivePolynomial][poly1,poly2] e.
   g ModuloMultplication[4][{1,2,1,3}][{3,3,0},{3,3,0}] = {1,0}"

RingPower::usage = "RingPower[RingModulo][PrimitiveElement][Element] e.g RingPower[
   4][{1,0,0,3,2,3}][{1,0,0}] = {{1,0,0},{1,0,0,0,0},{1,2,1,0},{2,1,1,
   2,1},{1,0,2,0,1},{2,1,3,1,0},{3,3,1,0,1},{1,3,2,1,3},{2,2,0,3,
   3},{1,1,2,2},{1,2,3,2,1},{3,3,1,1,2},{1,0,3,1,3},{3,2,1,1,
   0},{1,0,0,3,2},{1,0},{1,0,0,0},{1,2,1},{1,2,1,0,0},{1,1,0,
   1,2},{2,1,3,1},{1,3,3,0,2},{3,1,3,3,3},{3,2,2,1,1},{2,0,1,
   3,2},{1,1,2,2,0},{2,3,3,3,1},{3,1,0,0,3},{3,2,1,1},{2,1,0,2,3},{1}}"

CyclicMultiplativeGroup::usage = "CyclicMultiplativeGroup[ModuloRing][
   PrimitivePolynomial] e.g. CyclicMultiplativeGroup[4][{1,2,1,3}] = {{1},{1,
   0},{1,0,0},{2,3,1},{3,3,2},{1,3,3},{1,2,1}} and  CyclicMultiplativeGroup[
   4][{1,0,0,3,2,3}] = {{1},{1,0,0},{1,0,0,0,0},{1,2,1,0},{2,1,1,2,1},
   {1,0,2,0,1},{2,1,3,1,0},{3,3,1,0,1},{1,3,2,1,3},{2,2,0,3,3},{1,1,2,
   2},{1,2,3,2,1},{3,3,1,1,2},{1,0,3,1,3},{3,2,1,1,0},{1,0,0,3,2},
   {1,0},{1,0,0,0},{1,2,1},{1,2,1,0,0},{1,1,0,1,2},{2,1,3,1},
   {1,3,3,0,2},{3,1,3,3,3},{3,2,2,1,1},{2,0,1,3,2},{1,1,2,2,
   0},{2,3,3,3,1},{3,1,0,0,3},{3,2,1,1},{2,1,0,2,3}}"

ZeroSequences::usage =
 "ZeroSequences[ModuloRing][Order] generates the zero ideal ZeroSequences[4][
   3]= {{0,0,0},{0,0,2},{0,2,0},{0,2,2},{2,0,0},{2,0,2},{2,2,0},{2,2,2}}"

ZeroPad::usage = "ZeroPad[Order][Element] eg ZeroPad[3][{1}] = {0,0,1}"
```

```
TwoAdicExpansion::usage = "TwoAdicExpansion[ModuloRing][PrimitivePolynomial] e.g
   TwoAdicExpansion[4][{1,2,1,3}] = {{{0,0,0},{0,0,0},{}},{{0,0,0},{0,0,1},
   {2}},{{0,0,0},{0,1,0},{2,0}},{{0,0,0},{1,0,0},{2,0,0}},{{0,0,0},{2,3,
   1},{2,2}},{{0,0,0},{3,3,2},{2,2,0}},{{0,0,0},{1,3,3},{2,2,2}},{{0,0,0},
   {1,2,1},{2,0,2}},{{0,0,1},{0,0,0},{1}},{{0,0,1},{0,0,1},{3}},{{0,0,1},
   {0,1,0},{2,1}},{{0,0,1},{1,0,0},{2,0,1}},{{0,0,1},{2,3,1},{2,3}},{{0,0,
   1},{3,3,2},{2,2,1}},{{0,0,1},{1,3,3},{2,2,3}},{{0,0,1},{1,2,1},{2,0,
   3}},{{0,1,0},{0,0,0},{1,0}},{{0,1,0},{0,0,1},{1,2}},{{0,1,0},{0,1,0},
   {3,0}},{{0,1,0},{1,0,0},{2,1,0}},{{0,1,0},{2,3,1},{3,2}},{{0,1,0},{3,3,
   2},{2,3,0}},{{0,1,0},{1,3,3},{2,3,2}},{{0,1,0},{1,2,1},{2,1,2}},{{1,0,
   0},{0,0,0},{1,0,0}},{{1,0,0},{0,0,1},{1,0,2}},{{1,0,0},{0,1,0},{1,2,
   0}},{{1,0,0},{1,0,0},{3,0,0}},{{1,0,0},{2,3,1},{1,2,2}},{{1,0,0},{3,3,
   2},{3,2,0}},{{1,0,0},{1,3,3},{3,2,2}},{{1,0,0},{1,2,1},{3,0,2}},{{2,3,
   1},{0,0,0},{2,3,1}},{{2,3,1},{0,0,1},{2,3,3}},{{2,3,1},{0,1,0},{2,1,
   1}},{{2,3,1},{1,0,0},{3,1}},{{2,3,1},{2,3,1},{2,1,3}},{{2,3,1},{3,3,2},
   {1,1}},{{2,3,1},{1,3,3},{1,3}},{{2,3,1},{1,2,1},{3,3}},{{3,3,2},{0,0,
   0},{3,3,2}},{{3,3,2},{0,0,1},{3,3,0}},{{3,3,2},{0,1,0},{3,1,2}},{{3,3,
   2},{1,0,0},{1,3,2}},{{3,3,2},{2,3,1},{3,1,0}},{{3,3,2},{3,3,2},{1,1,
   2}},{{3,3,2},{1,3,3},{1,1,0}},{{3,3,2},{1,2,1},{1,3,0}},{{1,3,3},{0,0,
   0},{1,3,3}},{{1,3,3},{0,0,1},{1,3,1}},{{1,3,3},{0,1,0},{1,1,3}},{{1,3,
   3},{1,0,0},{3,3,3}},{{1,3,3},{2,3,1},{1,1,1}},{{1,3,3},{3,3,2},{3,1,
   3}},{{1,3,3},{1,3,3},{3,1,1}},{{1,3,3},{1,2,1},{3,3,1}},{{1,2,1},{0,0,
   0},{1,2,1}},{{1,2,1},{0,0,1},{1,2,3}},{{1,2,1},{0,1,0},{1,0,1}},{{1,
   2,1},{1,0,0},{3,2,1}},{{1,2,1},{2,3,1},{1,0,3}},{{1,2,1},{3,
   3,2},{3,0,1}},{{1,2,1},{1,3,3},{3,0,3}},{{1,2,1},{1,2,1},{3,2,3}}}"

σ::usage = "σ[ModuloRing][PrimitivePolynomial][TwoadicElement] e.g "

AutomorphismSigma::usage = "AutomorphismSigma[ModuloRing][PrimitivePolynomial] e.g
   AutomorphismSigma[4][{1,2,1,3}] = {{{0,0,0},{0,0,0}},{{0,0,2},{0,0,2}},
   {{0,2,0},{2,0,0}},{{2,0,0},{2,2,0}},{{0,2,2},{2,0,2}},{{2,2,0},{0,2,
   0}},{{2,2,2},{0,2,2}},{{2,0,2},{2,2,2}},{{0,0,1},{0,0,1}},{{0,0,3},{0,
   0,3}},{{0,2,1},{2,0,1}},{{2,0,1},{2,2,1}},{{0,2,3},{2,0,3}},{{2,2,1},
   {0,2,1}},{{2,2,3},{0,2,3}},{{2,0,3},{2,2,3}},{{0,1,0},{1,0,0}},{{0,1,
   2},{1,0,2}},{{0,3,0},{3,0,0}},{{2,1,0},{3,2,0}},{{0,3,2},{3,0,2}},{{2,
   3,0},{1,2,0}},{{2,3,2},{1,2,2}},{{2,1,2},{3,2,2}},{{1,0,0},{3,3,2}},
   {{1,0,2},{3,3,0}},{{1,2,0},{1,3,2}},{{3,0,0},{1,1,2}},{{1,2,2},{1,3,
   0}},{{3,2,0},{3,1,2}},{{3,2,2},{3,1,0}},{{3,0,2},{1,1,0}},{{2,3,1},{1,
   2,1}},{{2,3,3},{1,2,3}},{{2,1,1},{3,2,1}},{{0,3,1},{3,0,1}},{{2,1,3},
   {3,2,3}},{{0,1,1},{1,0,1}},{{0,1,3},{1,0,3}},{{0,3,3},{3,0,3}},{{3,3,
   2},{0,1,0}},{{3,3,0},{0,1,2}},{{3,1,2},{2,1,0}},{{1,3,2},{2,
   3,0}},{{3,1,0},{2,1,2}},{{1,1,2},{0,3,0}},{{1,1,0},{0,3,2}},
   {{1,3,0},{2,3,2}},{{1,3,3},{2,3,1}},{{1,3,1},{2,3,3}},{{1,1,
   3},{0,3,1}},{{3,3,3},{0,1,1}},{{1,1,1},{0,3,3}},{{3,1,3},{2,
   1,1}},{{3,1,1},{2,1,3}},{{3,3,1},{0,1,3}},{{1,2,1},{1,3,3}},
   {{1,2,3},{1,3,1}},{{1,0,1},{3,3,3}},{{3,2,1},{3,1,3}},{{1,0,
   3},{3,3,1}},{{3,0,1},{1,1,3}},{{3,0,3},{1,1,1}},{{3,2,3},{3,1,1}}}"

UnitsRing::usage = "UnitsRing[ModuloRing][PrimitivePolynomial] e.g. UnitsRing[4][
   {1,2,1,3}] = {{0,0,1},{0,0,3},{0,1,0},{0,1,1},{0,1,2},{0,1,3},{0,2,
   1},{0,2,3},{0,3,0},{0,3,1},{0,3,2},{0,3,3},{1,0,0},{1,0,1},{1,0,2},{1,
   0,3},{1,1,0},{1,1,1},{1,1,2},{1,1,3},{1,2,0},{1,2,1},{1,2,2},{1,2,3},
   {1,3,0},{1,3,1},{1,3,2},{1,3,3},{2,0,1},{2,0,3},{2,1,0},{2,1,1},{2,1,
   2},{2,1,3},{2,2,1},{2,2,3},{2,3,0},{2,3,1},{2,3,2},{2,3,3},{3,
   0,0},{3,0,1},{3,0,2},{3,0,3},{3,1,0},{3,1,1},{3,1,2},{3,1,
   3},{3,2,0},{3,2,1},{3,2,2},{3,2,3},{3,3,0},{3,3,1},{3,3,2},{3,3,3}}"

𝒯::usage = "𝒯[Order_][σ_][x_] e.g 𝒯[4][x3][{3, 0, 3}] = "
```

```
TraceRepresentation::usage = "TraceRepresentation[ModuloRing_][
   PrimitivePolynomial_] e.g. TraceRepresentation[4][{1,2,1,3}] = {{{0,0,0},{0,
   0,0}},{{0,0,2},{0,0,0}},{{0,2,0},{2,2,0}},{{2,0,0},{0,2,0}},{{0,2,2},
   {2,2,0}},{{2,2,0},{2,0,0}},{{2,2,2},{2,0,0}},{{2,0,2},{0,2,0}},{{0,0,
   1},{0,0,2}},{{0,0,3},{0,0,2}},{{0,2,1},{2,2,2}},{{2,0,1},{0,2,2}},{{0,
   2,3},{2,2,2}},{{2,2,1},{2,0,2}},{{2,2,3},{2,0,2}},{{2,0,3},{0,2,2}},
   {{0,1,0},{1,1,0}},{{0,1,2},{1,1,0}},{{0,3,0},{3,3,0}},{{2,1,0},{1,3,
   0}},{{0,3,2},{3,3,0}},{{2,3,0},{3,1,0}},{{2,3,2},{3,1,0}},{{2,1,2},{1,
   3,0}},{{1,0,0},{0,3,2}},{{1,0,2},{0,3,2}},{{1,2,0},{2,1,2}},{{3,0,0},
   {0,1,2}},{{1,2,2},{2,1,2}},{{3,2,0},{2,3,2}},{{3,2,2},{2,3,2}},{{3,0,
   2},{0,1,2}},{{2,3,1},{3,1,2}},{{2,3,3},{3,1,2}},{{2,1,1},{1,3,2}},{{0,
   3,1},{3,3,2}},{{2,1,3},{1,3,2}},{{0,1,1},{1,1,2}},{{0,1,3},{1,1,2}},
   {{0,3,3},{3,3,2}},{{3,3,2},{3,0,2}},{{3,3,0},{3,0,2}},{{3,1,2},{1,2,
   2}},{{1,3,2},{3,2,2}},{{3,1,0},{1,2,2}},{{1,1,2},{1,0,2}},{{1,1,0},{1,
   0,2}},{{1,3,0},{3,2,2}},{{1,3,3},{3,2,0}},{{1,3,1},{3,2,0}},{{1,1,
   3},{1,0,0}},{{3,3,3},{3,0,0}},{{1,1,1},{1,0,0}},{{3,1,3},{1,
   2,0}},{{3,1,1},{1,2,0}},{{3,3,1},{3,0,0}},{{1,2,1},{2,1,0}},
   {{1,2,3},{2,1,0}},{{1,0,1},{0,3,0}},{{3,2,1},{2,3,0}},{{1,0,
   3},{0,3,0}},{{3,0,1},{0,1,0}},{{3,0,3},{0,1,0}},{{3,2,3},{2,3,0}}}"

TupleRepresentation::usage = "TupleRepresentation[ModuloRing][PrimitivePolynomial]
   designed for fields. e.g.TupleRepresentation[2][{1,0,0,1,0,1}] = {{0,0,0,
   0,1},{0,0,0,1,0},{0,0,1,0,0},{0,1,0,0,0},{1,0,0,0,0},{0,0,1,0,1},
   {0,1,0,1,0},{1,0,1,0,0},{0,1,1,0,1},{1,1,0,1,0},{1,0,0,0,1},{0,0,
   1,1,1},{0,1,1,1,0},{1,1,1,0,0},{1,1,1,0,1},{1,1,1,1,1},{1,
   1,0,1,1},{1,0,0,1,1},{0,0,0,1,1},{0,0,1,1,0},{0,1,1,0,0},
   {1,1,0,0,0},{1,0,1,0,1},{0,1,1,1,1},{1,1,1,1,0},{1,1,0,0,
   1},{1,0,1,1,1},{0,1,0,1,1},{1,0,1,1,0},{0,1,0,0,1},{1,0,0,1,0}}"

MinimumPoly::usage = "MinimumPoly[ModuloRing][PrimitivePolynomial][rootPower][
   var]  designed for fields.e.g. MinimumPoly[2][{1,0,0,1,0,1}][3][x] "

GraeffeMethod::usage = "GraeffeMethod[GF2minpoly] generates a primitive
   polynomial in rung z4 e.g GraeffeMethod[{1,0,0,1,0,1}] = {1,0,0,3,2,3}  "

SequenceGenerator::usage =
 "SequenceGenerator[ModuloRing][PrimitivePoly][InitialCond][SeqLength]
   e.g. SequenceGenerator[4][{1, 0, 0, 3, 2, 3}][{0, 0, 0, 0, 2}][31] =
  {{2, 0, 0, 0, 0, 2, 0, 0, 2, 0, 2, 2, 0,
     0, 2, 2, 2, 2, 2, 0, 0, 0, 2, 2, 0, 2, 2, 2, 0, 2, 0}, {0, 0, 0, 0, 2}}
   "

InitialConditions::usage = "InitialConditions[PrimitivePoly] e.g InitialConditions[
   {1,0,0,3,2,3}] = {{0,0,0,0,1},{0,0,0,0,3},{0,0,0,2,1},{0,0,0,2,
   3},{0,0,2,0,1},{0,0,2,0,3},{0,0,2,2,1},{0,0,2,2,3},{0,2,0,
   0,1},{0,2,0,0,3},{0,2,0,2,1},{0,2,0,2,3},{0,2,2,0,1},{0,2,
   2,0,3},{0,2,2,2,1},{0,2,2,2,3},{2,0,0,0,1},{2,0,0,0,3},{2,
   0,0,2,1},{2,0,0,2,3},{2,0,2,0,1},{2,0,2,0,3},{2,0,2,2,1},
   {2,0,2,2,3},{2,2,0,0,1},{2,2,0,0,3},{2,2,0,2,1},{2,2,0,2,
   3},{2,2,2,0,1},{2,2,2,0,3},{2,2,2,2,1},{2,2,2,2,3},{0,0,0,0,2}} "

GoldSequence::usage = "GoldSequence[PrimitivePoly][InitCondNumber] e.g.
   GoldSequence[{1,0,0,3,2,3}][33] = {1,0,0,0,0,1,0,0,1,0,1,
   1,0,0,1,1,1,1,1,0,0,0,1,1,0,1,1,1,0,1,0} "

AutocorrelationSequence::usage = "AutocorrelationSequence[GoldSequence] e.g.
   AutocorrelationSequence[{1,0,0,0,0,1,0,0,1,0,1,1,0,0,1,1,1,1,1,0,
   0,0,1,1,0,1,1,1,0,1,0}] = {31,-1,-1,-1,-1,-1,-1,-1,-1,-1,
   -1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1} "

SpecifiedGoldSequences::usage = "SpecifiedGoldSequences[PrimitivePoly_][
   NumberList_] e.g. SpecifiedGoldSequences[{1,0,0,3,2,3}][{1,2,3,Last -
   1,Last}] = {{0,0,0,0,0,0,0,0,0,0,1,0,0,0,1,0,1,1,0,1,1,1,
   0,0,0,1,1,1,0,0,1,1},{1,0,0,0,0,1,0,0,1,1,1,1,0,1,1,0,
   0,1,0,1,1,0,1,1,1,0,0,1,0,0,1},{0,1,0,0,0,0,0,1,0,0,0,0,
   1,1,1,0,0,0,1,0,0,1,0,1,0,1,1,1,0},{1,1,1,1,1,1,0,
   0,0,1,0,0,1,1,0,0,0,1,1,1,1,1,0,1,0,1,0,1,1,1,1,1},{1,
   0,0,0,0,1,0,0,1,0,1,1,0,0,1,1,1,1,1,0,0,0,1,1,0,1,1,1,0,1,0}}"
```

```
AllGoldSequences::usage = "AllGoldSequences[PrimitivePoly] e.g. AllGoldSequences[
    {1,0,0,3,2,3}] = {{0,0,0,0,0,0,0,0,0,1,0,0,0,1,0,1,1,0,1,1,1,0,
    0,0,1,1,1,0,0,1,1},{1,0,0,0,0,1,0,0,1,1,1,1,0,1,1,0,0,1,0,1,1,
    0,1,1,1,0,0,1,0,0,1},{0,1,0,0,0,0,1,0,0,0,0,1,1,1,0,0,0,1,0,0,
    1,0,0,1,0,1,0,1,1,1,0},{1,1,0,0,1,1,0,1,0,1,0,1,1,1,1,1,0,1,
    0,1,0,1,0,0,0,1,0,1,0,0},{0,0,1,0,0,1,0,1,1,1,0,1,1,0,1,0,1,0,
    1,0,0,0,1,1,0,1,0,0,1,1,1},{1,0,1,0,0,0,0,1,0,1,1,0,1,0,0,1,0,
    1,0,0,0,0,0,0,0,0,1,1,1,0,1},{0,1,1,0,0,1,1,1,1,0,0,0,0,0,1,1,
    0,1,0,1,0,0,1,0,1,1,1,1,0,1,0},{1,1,1,0,0,0,1,1,0,0,1,1,0,0,0,
    0,1,0,1,1,0,0,0,1,1,0,0,0,0,0,0},{0,0,0,1,0,0,1,0,1,0,0,0,1,0,
    1,0,0,0,1,1,0,1,0,1,0,0,1,1,0,0,1},{1,0,0,1,0,1,1,0,0,0,1,1,1,
    0,0,1,1,1,0,1,0,1,1,0,0,1,0,0,0,1,1},{0,1,0,1,0,0,0,0,1,1,0,1,
    0,0,1,1,1,1,0,0,0,1,0,0,1,0,0,0,1,0,0},{1,1,0,1,0,1,0,0,0,1,1,
    0,0,0,0,0,0,0,1,0,0,1,1,1,1,1,1,1,1,0},{0,0,1,1,0,1,1,1,0,0,
    0,1,0,1,0,1,0,0,1,0,1,1,1,0,1,0,0,1,1,0,1},{1,0,1,1,0,0,1,1,1,
    0,1,0,0,1,1,0,1,1,0,0,1,1,0,1,1,1,1,0,1,1,1},{0,1,1,1,0,1,0,1,
    0,1,0,0,1,1,0,0,1,1,0,1,1,1,1,1,0,0,1,0,0,0,0},{1,1,1,1,0,0,0,
    1,1,1,1,1,1,1,1,1,0,0,1,1,1,1,0,0,0,1,0,1,0,1,0},{0,0,0,0,1,0,
    0,1,0,0,1,0,0,0,1,0,0,1,1,1,1,1,1,0,0,0,0,0,1,1,0},{1,0,0,0,1,
    1,0,1,1,0,0,1,0,0,0,1,1,0,0,1,1,1,0,1,0,1,1,1,1,0,0},{0,1,0,0,
    1,0,1,1,0,1,1,1,1,0,1,1,1,0,0,0,1,1,1,1,1,0,1,1,0,1,1},{1,1,0,
    0,1,1,1,1,1,1,0,0,1,0,0,0,0,1,1,0,1,1,0,0,1,1,0,0,0,0,1},{0,0,
    1,0,1,1,0,0,1,0,1,1,1,1,0,1,0,1,1,0,0,1,0,1,1,0,1,0,0,1,0},{1,
    0,1,0,1,0,0,0,0,0,0,0,1,1,1,0,1,0,0,0,0,1,1,0,1,1,0,1,0,0,0},
    {0,1,1,0,1,1,1,0,1,1,1,0,0,1,0,0,1,0,0,1,0,1,0,0,0,0,0,1,1,1,
    1},{1,1,1,0,1,0,1,0,0,1,0,1,0,1,1,1,0,1,1,1,0,1,1,1,0,1,1,0,1,
    0,1},{0,0,0,1,1,0,1,1,1,1,1,0,1,1,0,1,1,1,1,1,0,0,1,1,1,1,0,1,
    1,0,0},{1,0,0,1,1,1,1,1,0,1,0,1,1,1,1,0,0,0,0,1,0,0,0,0,1,0,1,
    0,1,1,0},{0,1,0,1,1,0,0,1,1,0,1,1,0,1,0,0,0,0,0,0,0,0,1,0,0,1,
    1,0,0,0,1},{1,1,0,1,1,1,0,1,0,0,0,0,0,1,1,1,1,1,0,0,0,0,1,0,
    0,0,1,0,1,1},{0,0,1,1,1,1,1,0,0,1,1,1,0,0,1,0,1,1,1,0,1,
    0,0,0,0,1,1,1,0,0,0},{1,0,1,1,1,0,1,0,1,1,0,0,0,0,0,1,
    0,0,0,0,1,0,1,1,0,0,0,0,0,1,0},{0,1,1,1,1,1,0,0,0,0,1,
    0,1,0,1,1,0,0,0,1,1,0,0,1,1,1,0,0,1,0,1},{1,1,1,1,1,0,
    0,0,1,0,0,1,1,0,0,0,1,1,1,1,1,0,1,0,1,0,1,1,1,1,1},{1,
    0,0,0,0,1,0,0,1,0,1,1,0,0,1,1,1,1,1,0,0,0,1,1,0,1,1,1,0,1,0}}"

CodingTransform::usage = "CodingTransform[L][GoldSeq] e.g. We wish to optimise this
    signal when there is a bit transition. Therefore x (-1) in the x4BitState. It
    delivers the two transformed codes. one for each pulse is either 0 or 1
    CodingTransform[L][{1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0,
      1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 0, 0, 1,
      0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 1, 0, 1, 0, 1, 1,
      0, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0,
      1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1,
      0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1,
      0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0,
      0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 1, 1, 1, 1, 0,
      1, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0}] = {{I, -1,
      -I, 1, I, -1, -I, 1, -I, -1, I, -1, I, 1, I, -1, -I, 1, I, -1, I, -1, -I, -1, I, -1,
      I, -1, -I, 1, I, -1, I, 1, I, 1, I, 1, -I, -1, -I, 1, I, -1, I, -1, I, 1, -I, -1, -I,
      1, -I, 1, I, -1, I, 1, -I, 1, I, -1, I, -1, I, 1, I, -1, I, -1, -I, -1, -I, 1, -I, -1,
      I, -1, I, -1, I, 1, I, 1, -I, 1, -I, 1, I, 1, -I, -1, I, -1, I, 1, I, 1, -I, -1, -I,
      -1, -I, 1, I, 1, -I, 1, -I, -1, -I, 1, -I, -1, I, -1, -I, -1, -I, -1, I, -1, I, -1,
      I, -1, -I, -1, -I, -1, I, 1, I, 1, -I, -1, I, 1, I, 1, -I, -1, -I, 1, -I, -1, -I, 1,
      I, -1, I, 1, I, -1, I, -1, I, -1, I, -1, -I, 1, -I, 1, -I, 1, -I, -1, I, 1, -I, -1,
      -I, 1, -I, -1, I, 1, -I, 1, I, -1, -I, 1, -I, 1, -I, 1, I, -1, -I, -1, -I, 1, I, -1,
      I, 1, -I, -1, -I, 1, I, 1, -I, 1, I, -1, I, -1, -I, 1, -I, 1, I, 1, I, 1, I, -1, I, 1,
      I, -1, I, 1, I, 1, -I, -1, I, -1, I, -1, I, -1, I, -1, I, 1, I, -1, -I, -1, I, 1,
      -I, -1, I, 1, -I, 1, -I, 1, I}, {1, I, 1, I, -1, -I, 1, I, 1, I, 1, I, -1,
      -I, 1, -I, 1, I, -1, -I, -1, I, 1, -I, 1, I, -1, I, 1, I, -1, -I, -1, -I, 1, I, 1, I,
      -1, I, -1, I, -1, -I, -1, I, -1, -I, -1, I, -1, I, 1, -I, -1, -I, -1, -I, -1, -I, -1,
      -I, -1, I, -1, -I, 1, -I, -1, I, 1, -I, -1, I, 1, I, 1, I, -1, I, -1, -I, 1, I, -1,
      -I, 1, -I, -1, I, -1, I, 1, I, -1, -I, 1, I, -1, I, -1, -I, -1, I, -1, I, -1, -I, 1,
      I, -1, I, 1, I, 1, I, 1, -I, -1, -I, 1, I, -1, I, -1, I, 1, -I, -1, -I, I,
      -I, 1, I, -1, I, 1, -I, 1, I, -1, I, -1, I, 1, I, -1, I, -1, -I, -1, -I,
      1, -I, -1, I, -1, I, -1, I, 1, I, 1, -I, 1, -I, 1, I, 1, -I, -1, I, -1,
      I, 1, I, 1, -I, -1, -I, -1, -I, 1, I, I, -I, 1, -I, -1, -I, 1, -I, -1, I,
      -1, -I, -1, -I, -1, I, -1, I, -1, I, -1, -I, -1, -I, -1, I, 1, I, 1, -I,
      -1, I, 1, I, 1, -I, -1, -I, 1, -I, -1, -I, 1, I, -1, I, 1, I, -1, I, -1,
"     I, -1, I, -1, -I, 1, -I, 1, -I, 1, -I, -1, I, 1, -I, -1, -I, 1, -I, -1}}
```

```
CodingTransformNew::usage = "CodingTransformNew[L][GoldSeq] e.g. We wish to
    optimise this signal when there is a bit transition. Therefore  x(-1) in the
    x4BitState.  It delivers 4 transformed codes. one for each pulse is either 0
    or 1 and then for the second pulse CodingTransform[L][{1,0,0,0,0,0,0,0,0,1,
    1,1,0,1,1,0,0,0,0,0,0,0,1,0,0,1,1,0,1,0,0,0,0,0,0,1,1,0,1,0,1,1,1,
    0,0,0,0,1,0,1,1,1,1,0,0,1,0,0,0,0,1,1,1,0,0,0,1,0,1,1,1,0,0,1,0,0,
    1,0,0,1,1,1,0,1,0,1,1,0,1,1,0,1,0,0,1,1,1,1,0,1,1,0,1,1,1,0,1,
    0,0,0,1,1,0,1,1,0,0,1,1,1,0,0,1,0,1,1,0,1,0,1,0,0,1,0,1,1,1,0,
    1,1,1,1,1,0,1,1,1,0,0,1,1,1,0,0,0,0,1,1,0,0,1,0,1,0,1,0,0,0,1,0,
    1,0,1,1,1,1,1,1,0,0,1,1,1,1,1,0,0,0,0,0,0,1,0,1,0,0,0,0,1,0,0,0,
    0,1,1,1,1,0,0,0,1,1,0,0,0,1,0,0,0,1,0,0,1,0,1,0,0,1,1,0,0,1,1,
    0,1,1,1,1,0,1,0,1,0,1,1,0,0,0,0,1,1,1,1,1,1,1,0,1,0,0}] =
    {{I,-1,-I,1,I,-1,-I,1,-I,-1,I,-1,I,1,I,-1,-I,1,I,-1,I,-1,-I,-1,
    I,-1,I,-1,-I,1,I,-1,I,1,I,1,I,1,-I,-1,-I,1,I,-1,I,-1,I,1,-I,-1,
    -I,1,-I,1,I,-1,I,1,-I,1,I,-1,I,-1,I,1,I,-1,I,-1,-I,-1,-I,1,-I,
    -1,I,-1,I,-1,I,1,I,1,-I,1,-I,1,I,1,-I,-1,I,-1,I,1,I,-1,-I,
    -1,-I,1,I,1,-I,1,-I,-1,-I,1,-I,-1,I,-1,-I,-1,-I,-1,I,-1,I,-1,I,
    -1,-I,-1,-I,-1,I,1,I,1,-I,-1,I,1,I,1,I,1,-I,-1,-I,1,-I,-1,-I,1,I,-1,
    I,1,I,-1,I,-1,I,-1,I,-1,-I,1,-I,-1,I,-1,-I,1,-I,-1,I,1,I,-1,-I,
    -1,I,1,-I,1,I,-1,-I,1,-I,1,-I,1,I,-1,-I,-1,-I,1,I,-1,I,1,-I,-1,
    -I,1,I,1,-I,1,I,-1,I,-1,-I,1,-I,1,I,1,I,1,I,-1,I,1,I,-1,I,1,I,
    1,-I,-1,-I,1,-I,1,-I,1,I,-1,I,-1,-I,-1,-I,1,-I,-1,I,-1,I,1,-I,
    -1,I,-1,-I,-1,I,-1,-I,-1,I,-1,I,-1,I,-1,-I,-1,-I,1,-I,1,I,-1,
    -I,1,I,-1,-I,-1,I,-1,I,-1,-I,1,-I,1,I,1,-I,-1,-I,-1,I,1,I,-1,-I,
    1,-I,-1,-I,1,-I,1,I,-1,-I,-1,-I,-1,-I,-1,-I,-1,-I,1,-I,-1,I,-1,
    -I,1,I,-1,-I,-1,I,-1,I,-1,I,1,-I,-1,-I,-1,I,1,I,1,I,1,-I,-1,-I,1,
    -I,1,I,-1,-I,-1,I,-1,I,-1,-I,1,-I,1,I,-1,I,-1,-I,-1,I,-1,I,-1,I,
    I,-1,I,1,-I,-1,-I,1,I,-1,-I,1,I,1,I,1,I,-1,I,-1,-I,1,-I,1,-I,
    -1,I,-1,I,1,I,-1,-I,-1,I,-1,I,-1,-I,-1,I,-1,I,-1,I,-1,-I,1,-I,
    I,1,I,1,-I,-1,-I,1,I,-1,I,-1,I,1,-I,-1,-I,1,I,-1,I,-1,I,-1,I,
    I,-1,I,-1,I,1,I,-1,-I,-1,-I,1,-I,1,I,-1,I,-1,-I,-1,I,-1,I,-1,I,
    -I,1,-I,-1,I,-1,-I,-1,-I,-1,I,-1,I,-1,I,-1,-I,-1,-I,1,I,-1,I,
    -1,-I,1,-I,1,-I,1,-I,-1,I,1,-I,1,-I,1,-I,-1}, {1,-I,1,-I,-1,I,1,
    -I,1,-I,1,-I,-1,I,1,I,1,-I,-1,I,-1,-I,1,I,1,-I,-1,-I,1,-I,-1,I,
    -1,I,1,-I,1,-I,-1,-I,-1,-I,1,I,-1,-I,1,-I,-1,I,-1,I,1,-I,-1,I,
    -1,I,-1,I,-1,I,-1,-I,-1,I,1,I,-1,-I,-1,I,-1,I,1,-I,-1,-I,1,-
```

```
CrosscorrelationSequence::usage = "CrosscorrelationSequence[{GoldSeq1,GoldSeq2}]
    e.g. seqlist =SpecifiedGoldSequences[{1,3,2,0,1,1,1,1,1,1,1}][{1,2,3,
    4,5,6,7,8,Last -1,Last}];Outer[(CrosscorrelationSequence[{#1,#2}]//Union )&,
    seqlist,seqlist,1] = {{{-33,-1,31,1023},{-65,-33,-1,31,63},{-65,-33,-1,
    31,63},{-65,-33,-1,31,63},{-65,-33,-1,31,63},{-65,-33,-1,31,63},{-65,-33,
    -1,31,63},{-65,-33,-1,31,63},{-65,-33,-1,31,63},{-33,31}},{{-65,-33,-1,
    31,63},{-33,-1,31,1023},{-65,-33,-1,31,63},{-65,-33,-1,31,63},{-65,-33,
    -1,31,63},{-65,-33,-1,31,63},{-65,-33,-1,31,63},{-65,-33,-1,31,63},{-65,
    -33,-1,31,63},{-33,31}},{{-65,-33,-1,31,63},{-65,-33,-1,31,63},{-65,-33,
    -1,31,63,1023},{-33,-1,31},{-65,-33,-1,31,63},{-65,-33,-1,31,63},{-65,
    -33,-1,31,63},{-65,-33,-1,31,63},{-65,-33,-1,31,63},{-33,31}},{{-65,-33,
    -1,31,63},{-65,-33,-1,31,63},{-33,-1,31},{-65,-33,-1,31,63,1023},{-65,
    -33,-1,31,63},{-65,-33,-1,31,63},{-65,-33,-1,31,63},{-65,-33,-1,31,63},
    {-65,-33,-1,31,63},{-33,31}},{{-65,-33,-1,31,63},{-65,-33,-1,31,63},{-65,
    -33,-1,31,63},{-33,-1,31,1023},{-65,-33,-1,31,63},{-65,-33,-1,31,63},
    {-65,-33,-1,31,63},{-65,-33,-1,31,63},{-65,-33,-1,31,63},{-33,31}},{{-65,
    -33,-1,31,63},{-65,-33,-1,31,63},{-65,-33,-1,31,63},{-65,-33,-1,31,63},
    {-65,-33,-1,31,63},{-33,-1,31},{-65,-33,-1,31,63},{-33,31}},{{-65,-33,
    -1,31,63},{-65,-33,-1,31,63},{-65,-33,-1,31,63},{-65,-33,-1,31,63},{-65,
    -33,-1,31,63},{-65,-33,-1,31,63},{-65,-33,-1,31,63},{-65,-33,-1,31,
    63},{-65,-33,-1,31,63,1023},{-33,-1,31},{-65,-33,-1,31,63},{-33,31}},
    {{-65,-33,-1,31,63},{-65,-33,-1,31,63},{-65,-33,-1,31,63},{-65,-33,-1,
    31,63},{-65,-33,-1,31,63},{-65,-33,-1,31,63},{-33,-1,31},{-65,-33,-1,31,
    63,1023},{-65,-33,-1,31,63},{-33,31}},{{-65,-33,-1,31,63},{-65,-33,-1,31,
    63},{-65,-33,-1,31,63},{-65,-33,-1,31,63},{-65,-33,-1,31,63},{-33,
    -1,31,1023},{-33,31}},{{-33,31},{-33,31},{-33,31},{-33,31},{-33,
    31},{-33,31},{-33,31},{-33,31},{-33,31},{-1,1023}}} "

Begin["`Private`"]

RingDivision[ModuloRing_][{Denominator_, Numerator_}] :=
 Module[{x1, x2, x3, x4, x5, x6, Qx, Nx},
  Clear[x1];
  x1[i_, j_] := Mod[i j, ModuloRing];
  x2 = Table[x1[Denominator[[1]], j], {j, 0, ModuloRing - 1}];
  x3 = Complement[ Table[i, {i, 0, ModuloRing - 1}], x2];
  Qx = {};
  Nx = Numerator;
  NoelUpdate :=
    Module[{ux1, ux2, ux3, ux4},
    If[MemberQ[x3, Nx[[1]]], Throw[{Qx, Nx, Denominator, False}]];
    ux1 = Take[Nx, Length[Denominator]];
    ux2 = Position[x2, ux1[[1]] // Flatten // (#[[1]] - 1) &;
   ux3 = Mod[ux1 - ux2 Denominator, ModuloRing];
    Qx = Join[Qx, {ux2}];
   ux4 = Drop[Nx, Length[Denominator]];
   Nx = Join[Rest[ux3], ux4]];
 Catch[Table[ NoelUpdate, {i, 1, Length[Numerator] - Length[Denominator] + 1}];
   {Qx, Nx, Denominator, True}]
  ]

DropLeadingZeros[DenominatorSeq_] :=
  Module[{x1},
  x1[{}] := {};
  x1[DS_] /; DS[[1]] === 0 := x1[ Rest[DS]];
 x1[DS_] := DS; x1[ DenominatorSeq]
 ]

PossibleDivisors[ModuloRing_][Order_] :=
   Module[{x1, x2, x3, x4},
  x1 = Table[{x2[i], 0, ModuloRing - 1}, {i, 0, Order }];
 x3 = Table[x2[i], {i, 0, Order }];
x4 =  Apply[Table, Sequence[Join[{x3}, x1]]] // Flatten[#, Order]& // Rest;
Map[DropLeadingZeros, x4] ]
```

```
PolynomialMultiplication[ModuloRing_][poly1_, poly2_] :=
   Module[{x1, x2, x3, x4, x5, x6, x7, x8, x9},
    x1 = Length[poly1] + Length[poly2] - 1;
    x2 = Table[x3^i, {i, 0, x1 - 1}];
    x4 = (Take[x2, Length[poly1]] // Reverse) . poly1;
    x5 = (Take[x2, Length[poly2]] // Reverse) . poly2;
    x6 = x5 x4;
    x7 = Table[Coefficient[x6, x3^(x1 - i - 1)] // Mod[#, ModuloRing]&, {i, 0, x1 - 2}];
    x8 = Join[x7, {Mod[x6 /. x3 -> 0, ModuloRing]}] // DropLeadingZeros]
```

We define polynomial multiplication, modulo a primitive polynomial

```
ModuloMultplication[ModuloRing_][PrimitivePolynomial_][poly1_, poly2_] :=
 Module[{x1, x2, x3, x4},
   x1 = PolynomialMultiplication[ModuloRing][poly1, poly2];
  x2 = RingDivision[ModuloRing][{PrimitivePolynomial, x1}];
 x2[[2]] // DropLeadingZeros]

RingPower[RingModulo_][PrimitiveElement_][Element_] :=
  Module[{x1, st, NoelUpdate},
  x1 = Length[PrimitiveElement] - 1;
  st = {1};
  NoelUpdate :=
   st = ModuloMultplication[RingModulo][PrimitiveElement][ st, Element];
  Table[NoelUpdate, {i, 1, 2^x1 - 1}]]

CyclicMultiplativeGroup[ModuloRing_][PrimitivePolynomial_] :=
 Module[{x1, x2, x3, x4, x5, x6, x7, x8, x9},
  x1 = (Length[PrimitivePolynomial] + 1) / 2 // Floor;
  x2 = PossibleDivisors[ModuloRing][x1];
x3 = Map[RingDivision[ModuloRing][{#, PrimitivePolynomial}]]&, x2];
x4 = Select[x3, #[[4]] == True &];
 x5 = Select[x4, #[[2]] == {1} &] // Transpose // Join[#[[1]], #[[3]]]&;
 x6 = Select[x4, #[[2]] == {ModuloRing - 1} &] // Transpose // Join[#[[1]], #[[3]]]&;
x7 = Join[{{1}, {ModuloRing - 1}}, x5, x6] // Union;
x8 = Map[ModuloMultplication[ModuloRing][PrimitivePolynomial][#, #]&, x7] // Union;
x9 = RingPower[ModuloRing][PrimitivePolynomial][x8[[2]]] // RotateRight ]


ZeroSequences[ModuloRing_][Order_] :=
   Module[{x1, x2, x3, x4},
  x1 = Table[{x2[i], 0, 1}, {i, 1, Order}];
 x3 = Table[ModuloRing / 2 x2[i], {i, 1, Order}];
x4 = Apply[Table, Sequence[Join[{x3}, x1]]] // Flatten[#, Order - 1]& ]

ZeroPad[Order_][Element_] :=
 Module[{x1, x2, x3},
  x1 = Table[0, {i, 1, Order - Length[Element]}];
  x2 = Join[x1, Element]]

TwoAdicExpansion[ModuloRing_][PrimitivePolynomial_] :=
 Module[{x1, x2, x3, x4, x5, x6, x7, x8, x9},
  x1 = Length[PrimitivePolynomial] - 1;
 x2 = CyclicMultiplativeGroup[ModuloRing][PrimitivePolynomial];
 x3 = Map[ZeroPad[x1], x2];
 x4 = Join[{Table[0, {i, 1, x1}]}, x3];
 x5 = Function[x, Mod[x[[1]] + 2 x[[2]], ModuloRing]];
  x6 = Table[{x4[[i]], x4[[j]], x5[{x4[[i]], x4[[j]]}]} // DropLeadingZeros},
     {i, 1, Length[x4]}, {j, 1, Length[x4]}] //
    Flatten[#, 1]&]

σ[ModuloRing_][PrimitivePolynomial_][TwoadicElement_] :=
   Module[{x1, x2, x3, x4, x5, x6, x7},
  {x1, x2, x3} = TwoadicElement;
  x4 = Length[PrimitivePolynomial] - 1;
  x5 = ModuloMultplication[ModuloRing][PrimitivePolynomial][x1, x1] // ZeroPad[x4];
 x6 = ModuloMultplication[ModuloRing][PrimitivePolynomial][x2, x2] // ZeroPad[x4];
 Mod[2 x6 + x5, ModuloRing]]
```

```
AutomorphismSigma[ModuloRing_][PrimitivePolynomial_] :=
 Module[{x1, x2, x3, x4, x5, x6, x7, x8, x9},
  x1 = TwoAdicExpansion[ModuloRing][PrimitivePolynomial];
 x2 = Map[σ[ModuloRing][PrimitivePolynomial], x1];
 x3 = Map[ZeroPad[Length[PrimitivePolynomial] - 1], Transpose[x1][[3]]];
 {x3, x2} // Transpose]

UnitsRing[ModuloRing_][PrimitivePolynomial_] :=
 Module[{x1, x2, x3, x4},
  x1 = AutomorphismSigma[ModuloRing][PrimitivePolynomial];
 x2 = x1 // Transpose // #[[1]]&;
 x3 =  x2 // Mod[2 #, 4]& // Union;
 Complement[x2, x3]]

UnitsRing[4][{1, 0, 0, 3, 2, 3}];

T[Order_][σ_][x_] := (NestList[σ, x, Order - 2] // Apply[Plus, #]&)

TraceRepresentation[ModuloRing_][PrimitivePolynomial_] :=
 Module[{x1, x2, x3, x4, x5, x6, x7, x8, x9},
  x1 = Length[PrimitivePolynomial] - 1;
  x2 = AutomorphismSigma[ModuloRing][PrimitivePolynomial];
  Map[(x3[Evaluate[#[[1]]]] := Evaluate[#[[2]]])&, x2];
 x4 =  Map[{#, T[x1][x3][#]}&, Transpose[x2][[1]]] // Mod[#, ModuloRing]&

]

TupleRepresentation[ModuloRing_][PrimitivePolynomial_] := Module[{x1, x2, x3},
  x1 = Length[ PrimitivePolynomial];
  x2 = NestList[ModuloMultplication[ModuloRing][PrimitivePolynomial][{1, 0}, #]&,
    {1}, 2^(x1 - 1) - 2];
 x3 = Map[ZeroPad[x1 - 1], x2]]
```

This algorithm does not always yield the correct value for the Minimum polynomial.

```
MinimumPoly[ModuloRing_][PrimitivePolynomial_][rootPower_][var_] :=
  Module[{x1, x2, x3, x4, x5, x6, x7, x8, x9, x10, x11, x12, x13, x14, x15, tom},
    x1 = Length[ PrimitivePolynomial] - 1;
   x2 = 2^x1 - 1;
x3 =  NestList[Mod[# + #, x2]&, rootPower, x2];
x4 =  Position[x3, rootPower] // (#[[2, 1]] - 1)&;
 x6 = Take[x3, x4] ;
  x7 = TupleRepresentation[ModuloRing][PrimitivePolynomial];
x8 =  Table[i, {i, 0, x2 - 1}];
 x9 = {x8, x7} // Transpose;
  x10 = Map[(var  + tom[#])&, x6] // Apply[Times, #]& // Expand;
  rule  = tom[n_] tom[m_] -> tom[n + m];
  x11 = x10 //. rule;
 x12 = Table[ Coefficient[ x11, var, x1 - i], {i, 0, x1}];
 Map[(tom[Evaluate[#[[1]] ]] := Evaluate[#[[2]]] )& , x9];
 tom[n_]  := Evaluate[ tom[Mod[n, x2]]]; Mod[
x12, ModuloRing]]

GraeffeMethod[GF2minpoly_] :=
  Module[
  {x1, x2odd, x2even, x3odd, x3even, x4even, x4odd, x5, x6even, x6odd, x7, x8, x9},
  x1 = Length[GF2minpoly];
  x2odd = Table[{1, 0}, {i, 1, (x1 + 1) / 2 // Floor}] // Flatten // Take[#, -x1]&;
  x2even = Table[{0, 1}, {i, 1, (x1 + 1) / 2 // Floor}] // Flatten // Take[#, -x1]&;
 x3even = GF2minpoly x2even // DropLeadingZeros;
  x3odd = GF2minpoly x2odd // DropLeadingZeros;
 x4even = PolynomialMultiplication[4][x3even, x3even];
 x4odd = PolynomialMultiplication[4][x3odd, x3odd] ;
 x5 = Max[{Length[x4even], Length[x4odd]}];
 x6even = ZeroPad[x5][x4even];
  x6odd = ZeroPad[x5][x4odd];
x7 =  x6even - x6odd;
 x8 =  (x7 Sign[x7[[1]]]) // Mod[#, 4]&;
  Join[x8, {0}] // Partition[#, 2]& // Transpose[#][[1]]& ]
```

```
SequenceGenerator[ModuloRing_][PrimitivePoly_][InitialCond_][SeqLength_] :=
 Module[{x1, x2State, x3Update, x4},
  x1 = Mod[-Rest[PrimitivePoly], ModuloRing];
 x2State = InitialCond;
 x3Update := Module[{}, x4 = Last[x2State];
    x2State = Join[{Mod[x2State . x1, ModuloRing]}, Drop[x2State, -1]]; x4];
  {Table[x3Update, {i, 1, SeqLength}], x2State} ]

InitialConditions[PrimitivePoly_] := Module[{x1, x2, x3, x4, x5J, x6, x7},
  x1 = Length[PrimitivePoly] - 1;
   x2 = Table[{x3[i], 0, 1}, {i, 1, x1}]; x4 = Table[x3[i], {i, 1, x1}];
  x5J = (Table @@ Sequence[Join[{x4}, x2]]) // Flatten[#, x1 - 1]&;
 x6 = Mod[x5J[[1]] + 2 x5J[[2]], 4];
 x7 = Map[Mod[x5J[[2]] + 2 #, 4]&, x5J];
 Join[x7, {x6}]]

GoldSequence[PrimitivePoly_][InitCondNumber_] := Module[{x1, x2, x3, x4},
  x1 = Length[PrimitivePoly] - 1;
  x2 = InitialConditions[PrimitivePoly][[InitCondNumber]];
 x3 = SequenceGenerator[4][PrimitivePoly][x2][2^x1 - 1];
 x4 = x3[[1]] /. {0 → 0, 1 → 0, 2 → 1, 3 → 1} ]

AutocorrelationSequence[GoldSequence_] :=
   Module[{x1, x2State, x3Update, x4},
    x1 = GoldSequence /. 0 → -1;
 x2State = x1;
 x3Update := Module[{}, x4 = x1 . x2State; x2State = RotateRight[x2State]; x4];
  Table[x3Update, {i, 1, Length[GoldSequence]}] ]

SpecifiedGoldSequences[PrimitivePoly_][NumberList_] :=
 Module[{x1, x2, x3, x4},
  x1 = Length[PrimitivePoly] - 1;
x2 =  NumberList /. Last → 2^x1 + 1;
 Map[GoldSequence[PrimitivePoly][#]&, x2] ]

AllGoldSequences[PrimitivePoly_] :=
 Module[{x1, x2, x3, x4},
  x1 = Length[PrimitivePoly] - 1;
x2 =  Table[i, {i, 1, 2^x1 + 1}];
 Map[GoldSequence[PrimitivePoly][#]&, x2] ]

CrosscorrelationSequence[{GoldSeq1_, GoldSeq2_}] :=
   Module[{x1, x2State, x3Update, x4},
    x1 = GoldSeq1 /. 0 → -1;
 x2State = (GoldSeq2 /. 0 → -1);
 x3Update := Module[{}, x4 = x1 . x2State; x2State = RotateRight[x2State]; x4];
  Table[x3Update, {i, 1, Length[GoldSeq1]}] ]

CodingTransform[L_][GoldSeq_] :=
  Module[{x1, x2, x3State, x4BitState, x5AccState, x6Update, c1, c2, x7},
   x2 = GoldSeq /. 0 -> -1;
  x3State = -1;
  x4BitState = (-1) * Last[x2];
  x5AccState = 0;
 x6Update[x_] := Module[{}, x5AccState = x + x5AccState;
    x3State = x3State * (-1); c1 = x3State I^ x5AccState;
 c2 = x3State I^(x5AccState - x4BitState);
  x4BitState = x; {c1, c2} ];
 x7 = Map[x6Update, x2] // Transpose ];

CodingTransformNew[L_][GoldSeq_] :=
 Module[{x1, x2, x3State, x4BitState, x5AccState, x6Update, c1, c2, x7},
  x2 = GoldSeq /. 0 -> -1; x3 = CodingTrans[L][x2];
  x4 = CodingTrans[L][-x2];
  {x3[[1]], x4[[1]], x3[[2]], x4[[2]]}
 ]

CodingTrans[L_][BiPolarBitSeq_] := Module[{x1, x2, x3State, x4BitState,
   x5AccState, x6Update, c1, c2, x7}, x2 = BiPolarBitSeq; x3State = -1;
  x4BitState = - Last[x2]; x5AccState = 0; x6Update[x_] := Module[{},
    x5AccState = x + x5AccState; x3State = x3State*-1; c1 = x3State*I^x5AccState;
    c2 = x3State * I^(x5AccState - x4BitState); x4BitState = x; {c1, c2}];
  x7 = Transpose[x6Update /@ x2]]
```

```
End[]

EndPackage[]
```

```
BeginPackage["LaurentFunctions'"]

T::usage = "This is the symbol period"

BT::usage = "This is the usual product"

h::usage = "This is the raw gaussian pulse"

ϖ::usage = "This denotes modulation index Pi"

ψ::usage = "ψ[L,t] is Laurents function"

hFiltered::usage = "This is the filtered gaussian pulse"

PhaseAngle::usage = "This function takes some time to calculate. The following
   code will draw a graph of the functionPhaseFunction[t_] =N[ ϕL,t];
phasepoints =Table[{t,PhaseFunction[t T]}//N,{t,0,L ,1/40}];
ListPlot[phasepoints,PlotJoined -> True]"

PhaseAngleFast::usage = "PhaseAngleFast[L][t] speeds up the calculation
   of PhaseAngle by calculating PhaseAngle[L][t] with L numeric and t
   symbolic. It takes some time to calculate."

S::usage = "S = Sin[ϖ]"

J::usage = "J = e^(j ϖ)"

C::usage = "C = Cos[ϖ]"

M::usage = "M = 2^(L-1)"

ModulationIndex::usage = "ModulationIndex = h"

LaurentS::usage = "LaurentS[L][n][t] = Sin[ ψ[L,t + n T]]/ S"

LaurentC::usage = "LaurentC[L][K][t] is Laurents CK,t"

AlphaKI::usage = "AlphaKI[LL][K,i] is Laurents αK,i"

LaurentLK::usage = "LaurentLK[L][K] gives the support of CK,t"
```

The start of Modulator Definitions

```
ANKInitialStateSetUp::usage =
 "ANKInitialStateSetUp[L][K][InitBitSeq,AccumulatedPhase] sets up the sequence
   of prior states of AK,N that the modulator went thgough to get to the
   constelation point specified by  AccumulatedPhase which is really A0,0"

AKN::usage = "AKN[L][K][{State,AccumulatedPhase}] defines AK,N in terms of A0,N"

ModulatingPulse::usage =
 "The Pulse is assumed to have the following structure Pulse[L][K][t]"

NumberOfCurves::usage = "The number of pulses used by the modulator"

SamplingInterval::usage =
 "SamplingInterval is the interval between samples of the output of the modulator"

InitialState::usage = "InitialState is the set of bits that are assumed
   to be present before i.e {a-1,a-2, ...}"

StartingQuadrant::usage = "StartingQuadrant = A0,-1 and is a number"
```

```
Modulator::usage = "Modulator[L][BitSeq,Opts] assumes the following
   default options StartingQuadrant → 0,InitialState → Table[1,{i,
   1,20}],SamplingInterval → T/32,NumberOfCurves → 4,ModulatingPulse→
   LaurentC . The Pulse is assumed to have the following structure Pulse[L][K][t]"
```

Start of the Receiver Functions

```
FiltPulse::usage = "The default pulse and is called by FiltPulse[L][K][t]"

SyncSample::usage = "Given that the sampling interval is T/32,then sync
   sample has rang -16 to 16 and this moves the point used to demodulate "

Receiver::usage = " Receiver[L][InputSeq,Opts] assumes the following
   default options {StartingQuadrant→0,InitialState→{1,1,1,1,1,1,1,1,
   1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1},SamplingInterval -> T/32,ModulatingPulse →
   FiltPulse,SyncSample -> 0,NumberOfCurves -> 2    "

ReceiverProper::usage = "ReceiverProper[L][ StartingQuadrant, InitialState,
   SamplingInterval,ModulatingPulse,SyncSample,NumberOfCurves,InputSeq]
   is called by Receiver after all the options have been resolved"

Begin["`Private`"]
```

$$\sigma := \frac{\sqrt{\text{Log}[2]}}{2\,\pi\,BT}$$

$$h[t\_] := \frac{\text{Exp}\left[-\frac{t^2}{2\,\sigma^2\,T^2}\right]}{\sqrt{2\,\pi}\,\sigma\,T}$$

Ideally we would and did define the effect of the convolution of h[t] by the formula below. However, this version of *Mathematica* gives an error.

$$h_{Fil}[t\_] := \text{Release}\left[\text{Module}\left[\{\tau\},\ \int_{-\frac{T}{2}}^{\frac{T}{2}} \frac{h[t-\tau]}{T}\,d\tau\right]\right]$$

```
hFiltered[PulseWidth_][t_] := Module[{x1, x2, x3}, x1 =
    (N[#1, 40]&)[Table[{t1, ∫_(-T/2)^(T/2) h[t1 - τ]/T dτ}, {t1, -PulseWidth/2, PulseWidth/2, T/20}]];
  x2 = Interpolation[x1]; x2[t]]

Φ := N[ModulationIndex π]

C := Cos[Φ];
S := Sin[Φ];
J := (e^(I Φ) // Chop);
M := 2^(L-1);

PhaseAngle[L_][t_] /; t ≤ 0 := 0

PhaseAngle[L_][t_] /; t ≥ L T := Φ

PhaseAngle[L_][t_] :=
 PhaseAngle[L][t_] = Module[{x1, x2, x3, x4, x5, x6}, x1 = hFiltered[3 L T][t1 - L T/2];
   x2 = Table[{t2, Φ ∫_(-L T)^t2 Evaluate[x1] dt1}, {t2, 0, L T, T/100}]; Interpolation[x2][t]]
```

We need to ensure that we always culculate PhaseAngle with t symbolic first to do the calculation only once

```
PhaseAngleFast[L_][t_] := Module[{x1}, x1[tt_] = PhaseAngle[L][tt]; x1[t]];
```

```
ψ[L_, t_] /; 0 < t < LT := PhaseAngleFast[L][t]

ψ[L_, t_] /; 2LT > t ≥ LT := π - PhaseAngleFast[L][t - LT]
```

We need to put this to avoid the function being extrapolated

```
ψ[L_, t_] /; ! (0 < t < LT) && ! (2LT > t ≥ LT) := 0

LaurentS[L_][n_][t_] := Sin[ψ[L, t + nT]] / S

AlphaKI[LL_][K_, i_] /; (0 < i < LL) && (0 <= K < 2^(LL-1)) :=
 Module[{x1, x2, x3, KNum}, x1 := (x2 = Mod[KNum, 2]; KNum = (KNum - x2)/2; x2);
  KNum = K; x3 = Table[x1, {ii, 0, LL - 1}]; x3[[i]]]

LaurentLK[L_][K_] :=
 Module[{x1}, x1 = Table[L (2 - AlphaKI[L][K, ii]) - ii, {ii, 1, L - 1}]; Min[x1]]

LaurentC[L_][K_][t_] /; 0 <= K < 2^L :=
 LaurentS[L][0][t] ∏_{ii=1}^{L-1} LaurentS[L][ii + L AlphaKI[L][K, ii]][t]
```

The start of the modulator function

```
ANKInitialStateSetUp[L_][K_][InitBitSeq_, AccumulatedPhase_] :=
          Module[{x1, x2, x3, x4, x5, acuphase, initbitseq},
   initbitseq = InitBitSeq;
   acuphase = AccumulatedPhase;
    UpdateSeq :=
    Module[{},  x1 = acuphase - Sum[initbitseq[[i]] AlphaKI[L][K, i], {i, 1, L - 1}];
     acuphase =  acuphase - First[initbitseq]; initbitseq = Rest[ initbitseq];  x1];
   Table[ UpdateSeq, {i, 1, LaurentLK[L][K]}]]

AKN[L_][K_][{State_, AccumulatedPhase_}] :=
 AccumulatedPhase - Sum[State[[i + 1]] AlphaKI[L][K, i], {i, 1, L - 1}]

Options[Modulator] := {StartingQuadrant → 0, InitialState → Table[1, {i, 1, 20}],
   SamplingInterval → T / 32, NumberOfCurves → 4, ModulatingPulse → LaurentC}

Modulator[L_][BitSeq_, Opts___] :=
 Module[
   {x1, x2, x3, x4, x5, x6, state, AccumalatedPhase, seq, AKNState, Curves, Pulse},
x1 = SamplingInterval /. {Opts} /. Options[Modulator];
state = InitialState /. {Opts} /. Options[Modulator];
x3 = StartingQuadrant /. {Opts} /. Options[Modulator];
x4 = SamplingInterval /. {Opts} /. Options[Modulator];
Pulse = ModulatingPulse /. {Opts} /. Options[Modulator];
Curves = (NumberOfCurves /. {Opts} /. Options[Modulator] ) - 1;
AccumulatedPhase = x3 ;
seq = BitSeq;
Table[
    AKNState[K] = ANKInitialStateSetUp[L][K][state, AccumulatedPhase], {K, 0, Curves}];
x5 := Module[{}, state = Join[{First[seq]}, Drop[state, -1]];
AccumulatedPhase = AccumulatedPhase + First[seq];
seq = Rest[seq];
Table[AKNState[K] = Join[{AKN[L][K][{state, AccumulatedPhase}]},
       Drop[AKNState[K], -1]], {K, 0, Curves}];
x6[t_] = Sum[Sum[ (J)^AKNState[K][[i + 1]] Pulse[L][K][t + i T],
       {i, 0, LaurentLK[L][K] - 1}], {K, 0, Curves}];
Table[x6[t], {t, 0, T - x4, x4}]];
 Table[x5, {kk, 1, Length[BitSeq]}] // Flatten]

Options[Receiver] := {StartingQuadrant → 0,
  InitialState → {1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1}, SamplingInter
  ModulatingPulse → FiltPulse, SyncSample -> 0, NumberOfCurves -> 2};
```

```
Receiver[L_][InputSeq_, Opts___] :=
  Module[{x1, x2, x3, x4, x5, x6},
   x1 = StartingQuadrant /. {Opts} /. Options[Receiver];
   x2 = InitialState /. {Opts} /. Options[Receiver];
   x3 = SamplingInterval /. {Opts} /. Options[Receiver];
   x4 = ModulatingPulse /. {Opts} /. Options[Receiver];
   x5 = SyncSample /. {Opts} /. Options[Receiver];
   x6 = NumberOfCurves /. {Opts} /. Options[Receiver];
  ReceiverProper[L][x1, x2, x3, x4, x5, x6, InputSeq]]

ReceiverProper[L_][StartingQuadrant_, InitialState_, SamplingInterval_,
  ModulatingPulse_, SyncSample_, NumberOfCurves_, InputSeq_] :=
    Module[{x1, sgn, ReceivedSeq, ExpectedValue, seq, ReceiveNext, 𝔻, ℜ},
   x1 = T / SamplingInterval;
   ReceivedSeq = Partition[InputSeq, x1] // Transpose // #[[SyncSample + 1]]&;
   ExpectedValue =
    ModulatingPulse[L][0][(LaurentLK[L][0] / 2) T + SyncSample SamplingInterval];
   ℜ = StartingQuadrant;
   sgn = 0;
   𝔻 = {};
   seq = ReceivedSeq;
   ReceiveNext :=
   Module[{x1, x2},
     x1 = ( (-1)^sgn ⅉ^ℜ First[seq]) // Im;
    If[Abs[x1 - ExpectedValue] <= Abs[x1 + ExpectedValue],
      𝔻 = Join[ 𝔻, {1}]; ℜ = ℜ + 1, 𝔻 = Join[ 𝔻, {-1}]; ℜ = ℜ - 1 ];
    seq = Rest[seq]; sgn = Mod[sgn + 1, 2]];
 Table[ReceiveNext, {i, 1, Length[ReceivedSeq]}];
  𝔻]

End[]

EndPackage[]
```

```
Needs["Utilities`Notation`"]

NotebookOpen["LaurentPalette.nb"];

NotebookOpen["LaurentPalette2.nb"];

Symbolize[Ã]

Notation[L_L_,K_ ⇔ LaurentLK[L_][K_]]

Pattern::patsym : First element in pattern Pattern[B, _] is not a symbol.

Symbolize[h_fil]

Notation[h_fil[PulseWidth_][t_] ⇔ hFiltered[PulseWidth_][t_]]

Notation[ϕ_L_,e_ ⇔ PhaseAngle[L_][t_]]

Notation[α_LL_,K_,ii_ ⇔ AlphaKI[LL_][K_, ii_]]

Needs["Calculus`LaplaceTransform`"]

Needs["bessel`"]

Needs["Convolve`"]
```

```
Needs["Calculus`LaplaceTransform`"];
Bessel`Private`InverseLaplaceTransform:=InverseLaplaceTransform;
BeginPackage["bessel`"]

BesselFilter::usage =
"BesselFilter[order][LowPass[CutoffFrequency(AngularFrequency)]][FrequencyResponse][s]
generates a filter with a 6dB point specified. To get a 3dB specification
use
BesselFilter[order][LowPass3dB[CutoffFrequency(AngularFrequency)]][FrequencyResponse][
s]   ....
To get from s plane to the f domain make the transformation s -> 2 Pi f I "

GroupDelay::usage = " GroupDelay[freq][s][angle] gives the group delay seconds at
      the specified frequency ... "
LowPass::usage = " Lowpass .."
LowPass3dB::usage = "To get a 3dB point for the Bessel filter "
FrequencyResponse::usage " Continuous"
ImpulseResponse::usage = " Impulse Resp"
DigitalFrequencyResponse::usage ="Digital Frequency Response "
FrequencyResponseFunction::usage="Functional form of Frequency Response function"
RequestedFrequency::usage = "Used in GroupDelay"
Begin["`Private`"]


d[n_][k_] := (2 n - k)!/ (2^(n - k) (k!)(n -k)!)

Bessel[n_][s_] := Sum[d[n][k] s^k, {k,0,n}]

(*----- Definition of Low Pass Filter  ----------*)
BesselFilter[order_][LowPass3dB[CutoffFrequency_]][FrequencyResponse][s_]:=
        Module[{ss,tempfilter,x1,x2,x3,Omega,freq3dB,ord},
    tempfilter[ord_][ss_] := d[ord][0]/Bessel[ord][ss];
    amplitude = tempfilter[order][I Omega ] tempfilter[order][-I Omega]//Simplify;
        freq3dB = Select[Omega /. N[Solve[amplitude == 1/2,Omega]],Abs[#] == #&][[1]];
        tempfilter[order][s/CutoffFrequency freq3dB] ];
(*BesselFilter[order_][LowPass3dB[CutoffFrequency_]][FrequencyResponse][s_]:=
        Module[{ss,tempfilter,x1,x2,x3,Omega,freq3dB,ord},
    tempfilter[ord_][ss_] = d[ord][0]/Bessel[ord][ss];
    amplitude = Simplify[ComplexExpand[tempfilter[order][I
Omega ]Conjugate[tempfilter[order][I Omega]] ]];
    x3 = Numerator[amplitude];
    x1 = Denominator[amplitude];
    x2 = Simplify[ComplexExpand[x1/. Power[Abs[x_],4] -> Power[Re[x]^2 +
Im[x]^2,2]]];
    amplitude = Simplify[x3/x2];
        freq3dB = Select[Omega /. N[Solve[amplitude == 1/2,Omega]],Abs[#] == #&][[1]];
        tempfilter[order][s/CutoffFrequency freq3dB] ];*)

BesselFilter[order_][LowPass3dB[CutoffFrequency_]][ImpulseResponse][t_]:=
Module[{s,tt,x1},
x1 = N[BesselFilter[order][LowPass3dB[CutoffFrequency]][FrequencyResponse][s]];
InverseLaplaceTransform[x1,s,t]];


BesselFilter[order_][LowPass[CutoffFrequency_]][FrequencyResponse][s_]:=
        Block[{ss,tempfilter},
            tempfilter =  d[order][0]/ Apply[Times,
                                (s - ss/(d[order][0]^(1/order))) /.
                            N[Solve[Bessel[order][ss] == 0, ss]]];
            tempfilter/. s -> s/CutoffFrequency
             ]/; 0<= order < 15

(*------ Definition of BandPass Filter --------*)

BesselFilter[order_][BandPass[Lower_,Upper_]][FrequencyResponse][s_]:=
        Block[{ss,tempfilter},
            tempfilter =  d[order][0]/ Apply[Times,
                                (s - ss/(d[order][0]^(1/order))) /.
                            N[Solve[Bessel[order][ss] == 0, ss]]];
            tempfilter/. s -> (s^2 + Lower Upper)/(s (Upper - Lower))
             ]/; 0<= order < 15


BesselFilter[order_][LowPass[CutoffFrequency_]][ImpulseResponse][t_] :=
        Block[{ss, tempfilter,denomfilter,numfilter,roots},
            tempfilter =        (s - ss/(d[order][0]^(1/order))) /.
                            N[Solve[Bessel[order][ss] == 0, ss]];
            tempfilter =  tempfilter/. s -> s/CutoffFrequency;
            tempfilter = Simplify[tempfilter];
```

```
(*   :Title:    *)

(*   :Authors:      *)

(*   :Summary: *)
(*   :Context:    Global`   *)


(*   :History: Started on 15 June  91
          Worked on*)

(*   :Functions: NoelConvolve     *)

(*   :Source:     *)

(*   :Warning:    *)

(*   :Mathematica Version:  1.2   *)

(*   :Limitation:   *)

(*
     :Discussion:
 the program calculates raised cosine functions in the time and frequency domains*)


(*   B E G I N      I N I T I A L I Z A T I O N   *)



(*   E N D      I N I T I A L I Z A T I O N   *)


(* Program *)

BeginPackage["Convolve`"]

NoelConvolve::usage = "NoelConvolve[a,b] performs a convolution. a is a vector, b is
is ristricted to length of a being greater than length of b.  e.g.
NoelConvolve[{a1,a2,a3,a4,a5},{{b1,c1},{b2,c2}}] "
TensorConvolve::usage = "TensorConvolve[a,b] performs a convolution. a is a vector,
TensorConvolve[{a1,a2,a3,a4,a5},{{b1,c1},{b2,c2}}]. See also NoelConvolve "

MapConvolve::usage =
"MapConvolve[input,filter] performs vector convolution only. It is memory efficient.

Begin["`Private`"]

NoelConvolve[first_,second_]:= Block[{x1,x2,x3},
     x1 = Length[second];
     x2 = Table[0,{i,1, x1-1}];
     x3 = Join[x2,first,x2];
     x4 = Partition[x3,x1,1];
Map[Reverse[#] . second&, x4]]   /; Length[first] >= Length[second]

MapConvolve[input_,filter_]:=
Module[{i,x1,x2,x3},
          x1 = Table[0,{i,1,Length[filter] }];
          x2[a_] := {x1 = Join[Drop[x1,1],{a}]};
          x3 = Reverse[filter];
          Map[(x3 . x2[#])&,Join[input,Table[0,{i,1,Length[filter]-1}] ] ]];


TensorConvolve[input_,filter_]:= Transpose[  Map[MapConvolve[input,#]&,Transpose[fil


End[]

Protect[NoelConvolve,TensorConvolve,MapConvolve]
Null
EndPackage[]
```

```
            numfilter = d[order][0];
            denomfilter = Apply[Times, tempfilter];
            roots = Apply[Join,Map[Solve[#,s]&,Map[# ==0&,tempfilter]]];
            Apply[Plus, numfilter/D[denomfilter,s] E^(s t) /. roots]
];0 <= order < 15

BesselFilter[order_][BandPass[Lower_,Upper_]][ImpulseResponse][t_] :=
        Block[{ss, tempfilter,denomfilter,numfilter,roots},
            tempfilter =        (s - ss/(d[order][0]^(1/order))) /.
                              N[Solve[Bessel[order][ss] == 0, ss]];
            tempfilter =  tempfilter/. s -> (s^2 + Lower Upper)/s/(Upper - Lower);
            tempfilter = Simplify[s tempfilter];
            numfilter = d[order][0] s^order;
            denomfilter = Apply[Times, tempfilter];
            roots = Apply[Join,Map[Solve[#,s]&,Map[# ==0&,tempfilter]]];
            Apply[Plus, numfilter/D[denomfilter,s] E^(s t) /. roots]
];0 <= order < 15


GroupDelay[FrequencyResponse_][s_][AngularFrequency_]:=
Block[{temps,tempw,t1,t2,t3,t4,tempx},
                    t1 = FrequencyResponse /. s -> temps;
                    t1 = t1 /. temps -> I tempw;
                    t1 = Together[t1];
                    t2 = Expand[Denominator[t1]];
                    t2 = Distribute[Conjugate[t2]];
                    t2 = Map[Distribute[#, Times]&,t2];
                    t2 = t2 /. Conjugate[tempx_] -> tempx;
                    t2 = Expand[Numerator[t1] t2];
                    t3 = Distribute[Re[t2]];
                    t3 = Map[Distribute[#,Times]&,t3];
                    t3 = t3/. Re[x_] -> x;
                    t4 = ArcTan[Simplify[Expand[-I(t2 - t3)]]/t3];
                    t4 = -D[t4, tempw];
                    t4 /. tempw -> AngularFrequency
                    ]


GroupDelay::usage =
"GroupDelay[FrequencyResponseFunction,RequestedFrequency] gives the group delay
given the frequency response in terms of f at the requested frequency. Note the group
delay is usually evaluated at 0 for a lowpass filter.e.g if
 FrequencyResponsefParFunction =
Function[f,Evaluate[BesselFilter[6][LowPass3dB[100]][FrequencyResponse][2 Pi I f]]]
then GroupDelay[FrequencyResponseFunction,ff] can be ploted to show how wide the
linear region is."


GroupDelay[FrequencyResponseFunction_,RequestedFrequency_]:=
Module[{ f1,x1,x2,x3 },
x1 =
ArcTan[ComplexExpand[Im[FrequencyResponseFunction[f1]]/Re[FrequencyResponseFunction[f1
]]]];
x2 = Evaluate[-D[x1,f1]];
(x2/. f1 -> RequestedFrequency) /(2 Pi)];




(*--------Group Delay Calculations---------*)
(*
In[40]:=Table[N[%18],{w,2 Pi  990000,2 Pi 1010000,2 Pi 5000}]

Out[40]= {-0.00000277055, -0.00000275661, -0.00000274284, -0.00000272905,

>     -0.00000271467}

In[41]:= 1000000 2 Pi %40

Out[41]= {-5.5411 Pi, -5.51322 Pi, -5.48569 Pi, -5.4581 Pi, -5.42933 Pi}
```

```
In[42]:= %[[1]] - %[[5]]
Out[42]= -0.111769 Pi
In[43]:= % 360/(2 Pi)
Out[43]= -20.1184

*)
filter[s_] = BesselFilter[2][FrequencyResponse][s]

(* Digital Filter Design *)
(*------ Definition of Digital Low Pass Filter  ----------*)
BesselFilter[order_][LowPass[CutoffFrequency_]][DigitalFrequencyResponse[T_]][z_ ^
-1]:=
        Block[{s,ss,tempfilter,roots,sections,x},
       roots = N[Solve[Bessel[order][ss] == 0, ss]];
       sections = (s - ss/(d[order][0]^(1/order))) /. roots;
              sections = Partition[sections, 2];
              sections = Map[Apply[Times, #]&, sections];
       sections = Simplify[ sections];
              sections = d[order][0]^(2/order) /sections;
              sections = Map[ExpandAll[#]&, sections];
       sections = Map[N[Numerator[#]/d[order][0]^(2/order)] /
              N[ Denominator[#]/ d[order][0]^(2/order)]&, sections];
       sections = sections /. s -> s/N[CutoffFrequency];
       sections = sections /. s -> 2/T (1 - x)/( 1 + x );
              sections = Together[sections];
              sections = Map[ Expand[Numerator[#]/Denominator[#][[1]]] /
              Expand[Denominator[#]/Denominator[#][[1]]]&,sections];
        sections = sections /. x -> z^-1
             ]/; 0<= order < 15 && EvenQ[order]



(* Defining Auxilarry Functions *)
(* FilterSection = {{a[0],a[1], ... a[n]},{b[0],b[1], ...,b[m]}}
   Input is a vector of length n+1
   Feedback is a vector of length m + 1 *)

SingleOutput[FilterSection_][InputSection_][FeedbackSection_] :=
          FilterSection[[1]] . InputSection - Drop[FilterSection[[2]],1] .
FeedbackSection /;
                         FilterSection[[2]][[1]] == 1.

FormatFilter[QuadraticSection_][variable_] := Block[{i, acoefficients,
bcoefficients },
             acoefficients = Expand [Numerator[QuadraticSection] variable^2] ;
             const  = Apply[Plus,Table[variable^i ,{i,0,2}]];
             acoefficients = acoefficients + const;
             acoefficients = acoefficients /. acoefficients[[0]] -> List;
             acoefficients = acoefficients /. variable -> 1;
             acoefficients = acoefficients - 1;
             acoefficients = Reverse[acoefficients];
             bcoefficients = Expand [Denominator[QuadraticSection] variable^2] ;
             bcoefficients = bcoefficients + const;
             bcoefficients = bcoefficients /. bcoefficients[[0]] -> List;
             bcoefficients = bcoefficients /. variable -> 1;
             bcoefficients = bcoefficients - 1;
                      bcoefficients = Reverse[bcoefficients];
             {acoefficients,bcoefficients}]

Output[Section][FilterSection_,
AInitial_,BInitial_][InputSequence_][OutputSequence_]:= Block[{zeros, poles,
          inputsection,output,ainitial, binitial, outputsequence, inputsequence },
               zeros = Length[FilterSection[[1]]] - 1 ;
               poles = Length[FilterSection[[2]]] - 1;
               ainitial = Join[{First[InputSequence]},Drop[AInitial, -1]];
               output = SingleOutput[FilterSection][ainitial][BInitial];
               binitial = Join[{output},Drop[BInitial,-1]];
               outputsequence = Join[OutputSequence,{output}];
               inputsequence = Drop[InputSequence,1];
               Output[Section][FilterSection,
ainitial,binitial][inputsequence][outputsequence] ] /;
        InputSequence != {}
```

```
Output[Section][FilterSection_, AInitial_,
BInitial_ ][InputSequence_][OutputSequence_]:= OutputSequence /;
                              InputSequence == { }

Output[Filter_][InputSequence_] := InputSequence /; Filter == {}
Output[Filter_][InputSequence_] := Output[Drop[Filter,
1]][Apply[Output[Section],First[Filter]][InputSequence][{}]] /;
                              Filter != {}

Output2[Section][FilterSection_, AInitial_,BInitial_][InputSequence_]:=
            Block[{inputsection,output,ainitial, binitial, outputsequence,
inputsequence },
                  ainitial = AInitial;
                  binitial = BInitial;
                  outputsequence = {};
                  inputsequence = InputSequence;
                        While[ inputsequence != {}, Block[{},
                  ainitial = Join[(First[inputsequence]),Drop[ainitial, -1]];
                  output = SingleOutput[FilterSection][ainitial][binitial];
                  binitial = Join[{output},Drop[binitial,-1]];
                  outputsequence = Join[outputsequence,{output}];
                  inputsequence = Drop[inputsequence,1]
                              ]
                  ];
                  outputsequence ]

Output1[Filter_][InputSequence_] := InputSequence /; Filter == {}
Output1[Filter_][InputSequence_] := Output1[Drop[Filter,
1]][Apply[Output2[Section],First[Filter]][InputSequence]] /;
                              Filter != {}


(* Tests

 <<../support/bessel.m
 <<../environment/env.m
       filter = BesselFilter[4][LowPass[135000 2 Pi]][DigitalFrequencyResponse[T
/16]][z ^ -1];
        frequencyresponse =Apply[Times, filter /. z -> E^(I f 2 Pi T/16 1000000)];
        Plot[20Log[10,Abs[frequencyresponse]],{f,0,2}]
        Plot[Arg[frequencyresponse],{f,0,2}]
       filter = Map[FormatFilter[#][z]&,filter];
       filter = Table[{filter[[k]],{0,0,0},{0,0}},{k,1,2}];

  Output1[filter][N[Table[Cos[100000 k T/16] + Cos[1000000 k T/ 16], {k, 0,10}]]];


roots = N[Roots[Bessel[4][s] == 0, s]]

argroots =  Map[Function[x,Distribute[x,Equal]],Distribute[Arg[roots],Or]]

absroots =  Map[Function[x,Distribute[x,Equal]],Distribute[Abs[roots],Or]]

Plot[20Log[10,Abs[filter[I x]]],{x,0,5000}]
*)
End[]
EndPackage[]
```